Minimal models vs. logic programming: the case of counterfactual conditionals

Schulz, K.

DOI
10.1080/11663081.2014.911537
Publication date
2014
Document Version
Author accepted manuscript
Published in
Journal of Applied Non-Classical Logics

Minimal models with three-valued logic: The case of counterfactual conditionals

1. Introduction

This paper is about non-monotonic logic and its applications. I will try to bring together two different formalisms developed to describe non-monotonic reasoning: the Minimal Models approach and Logic Programming. The Minimal Models approach is probably the most intuitive and general approach to non-monotonic reasoning and, therefore, it is also the approach that is most often applied, certainly in philosophy and linguistics. The fact that the approach is so general and flexible and can be fitted to various different applications also comes with a price: it is computational complex and, for this reason, hard to work with.

I will argue in this paper that for this reason we – philosophers, semanticists and others working on applications of non-monotonic reasoning – should consider switching to a more user-friendly technical tool: Logic Programming. Logic Programming is rarely used for applications outside of the Computational Sciences, because the price to be payed for the very attractive computational properties of this framework is its limited expressibility. However, this doesn't need to be a problem as long as we can express everything needed for the particular application at hand. As will be argued here, Logic Programming is sufficient to account at least for one specific application: the semantics of conditional sentences.

This is not the first paper that argues along these lines. The extensions of Logic Programming that will be discussed here come from (Stennings and van Lambalgen, 2005) and (Schulz, 2011). Also these authors focus on applying Logic Programming to the semantics of conditional sentences. However, so far there has been no systematic discussion of the relation between these two approaches. This is what the present paper hopes to provide. More concretely, we will present a way to combine both frameworks into one unified approach. Additionally, we will link the resulting framework of Logic Programming to the Minimal Models approach by showing how the first can be translated into the second.

2. Non-monotonicity and counterfactual conditionals

Non-monotonicity is generally understood as a property of a notion of inference/entailment between sentences. Let $\models$ be some notion of entailment. Then we say that this notion is *monotonic* if for all sentences A, B, C of the language for which the notion of entailment is defined it holds that if $A \models C$ then $A, B \models C$. A notion of entailment between sentences is called *non-monotonic* in case it is not monotonic, thus, in case we can find sentences A, B, C such that even though $A \models C$, it does not hold that $A, B \models C$. In other words, in case extra information can mean loss of inferences.

In the literature on counterfactual conditionals we can often find the claim that conditionals are (or behave) non-monotonic. In this case the notion of non-monotonicity is applied to a class of sentences instead of an inference relation. One might wonder what is actually meant by calling them non-monotonic. If we take a look at the literature, the picture that emerges is not homogeneous. There are basically two different ways in which counterfactuals are claimed to give rise to non-monotonicity. In the first case, and this is what we find most of the times, non-monotonicity is observed to apply to the internal semantic properties of counterfactuals. This observation presumes a very intuitive and popular approach to the truth conditions of counterfactuals (and conditionals in general) that links the truth of these sentences to some notion of entailment (see (1)).

(1) A counterfactual with antecedent A and consequent C is true (given some model) in case $A \models C$, where $\models$ is some notion of entailment.

If one adopts (1), then it can be easily argued that the relevant notion of entailment involved in (1) is non-monotonic. This can be illustrated with (2), following a famous example of (Goodman, 1955).

(2) a. If I strike this match, it will lit.
    b. If I strike this match and it is wet, it will lit.

If we agree that the first conditional is true while the second is false, and we accept that (1) gives a correct description of the truth conditions of conditionals, then with A for *I strike that match*, B for *that match is wet*, and C for *that match is lit*, we have $A \models C$, and $A, B \not\models C$. Thus, the notion of entailment on which (1) assumes the truth conditions of counterfactual to be based, appears to be non-monotonic.

There is also a different sense in which conditionals have been claimed

to involve non-monotonic reasoning. In this case non-monotonicity does not apply to the internal semantic structure of conditionals, but to reasoning with conditionals. The most famous example come from reasoning experiments in psychology (Byrne, 1989). Consider the four statements in (3).

(3) a. A_1 : If she has an essay to write, she will study late in the library.
b. A_2 : She has an essay to write.
c. C : She will study late in the library.
d. B : If the library is open, she will study late in the library.

Empirical research shows that 95% of the test population says that C follows from A_1 and A_2. With an extra premise B (3-d) this percentage drops to 60%. Again, we are confronted with an non-monotonic inference pattern, now in a probabilistic variation: the validity of an inference becomes less probable with an extra premise.

On first view the distinction between reasoning *within* conditionals (illustrated with (1)) and reasoning *with* conditionals (exemplified by (3)), which are then both observed to behave non-monotonic, appears artificial. It seems very obvious that in the end we are talking in both cases about the same notion of inference. Still, in the literature both cases are hardly discussed together. Furthermore, approaches that can deal with one of the two phenomena often do not easily extend to the other. Just to focus on two approaches discussed in this paper: (Stennings and van Lambalgen, 2005) aims to account for reasoning with conditionals, but cannot directly be applied to reasoning within conditionals, because this approach does not embrace (1). (Schulz, 2011), on the other hand, focusses on spelling out (1) in an adequate way, but is limited to reasoning within conditionals. The reason is that the notion of entailment proposed in (Schulz, 2011) cannot deal with premise sets that contain conditionals.

For the rest of the paper we will ignore the distinction between reasoning with or within conditionals. But because we will regularly switch between examples taking one or the other perspective, the reader should be aware of it.

Orthogonally to the discussion so far, there are also different types of non-monotonic inferences conditionals can give rise to. One type has been already illustrated by the examples distinguishing the two ways conditionals can be involved in non-monotonic reasoning. Discussing it in more detail will demand a bit more conceptual background.

Conditionals are known for standing in a very intimate relation to laws or generalizations. The latter are taken to be intrinsically related to the truth of conditionals (i.e. (2-a) is true *because* we assume a certain causal relation between striking a match a its lightening up)[1]. This can be reflected in a recipe like (1) by redefining $\models$ as depending on some set L of given laws of dependencies. In reasoning with conditionals, as exemplified in (3) this comes back in the interpretation of the conditional premisses: the first premise is understood as introducing a generalization about the behavior of the person the sentence talks about. In both cases the conclusion (the consequent of the conditional in (2) in the first case, and the sentence (3-c) in the second case) is derived based on this law or generalization. And also in both cases, the non-monotonistic behavior is provoked by extra information about an exception to this rule. When there is explicit information that we have to deal with an exception to a relevant law/generalization, the derivation of the conclusion using this law/generalization no longer goes through. So, this is one way in which conditionals might lead to non-monotonic reasoning: extra information about an exception to a law will stop the law from being applied.

There is also a different type of non-monotonic inference that can occur with conditionals. In this case the additional premise does not witnesses an exception to the rule, but it simply observes that the consequent of the rule does not hold. Example (4) illustrates this for reasoning with conditionals (though the same point can be made for reasoning within conditionals as well). In this example intuitively C follows from A_1 and A_2. If we add the observation B, the conclusion C is no longer plausible.

(4) a. A_1 : If you drop glass, it breaks.
b. A_2 : She dropped that wineglass.
c. C : It broke.
d. B : The glass didn't break.

Though the example in this form looks rather trivial, and might therefore be deemed uninteresting, the involved reasoning pattern is central to modeling, in particular counterfactual reasoning. To understand why, lets go back to (1). It is well accepted in the literature that in order to account for counterfactual conditionals, the scheme in (1) needs to be amended as in (5): for a counterfactual conditional to be true, the

[1] Similarly, it has been argued, for instance by Lewis, that there is a causal relation between striking a match and its lightening up *because* the conditional is true. We will not dive into this discussion here.

consequent must be entailed by the antecedent and a set of singular facts about the evaluation world (see, for instance, (Veltman, 2005)).

(5) A counterfactual with antecedent A and consequent C is true in world w_0 in case $A \cup B(w_0) \models_L C$, where $B(w_0)$ is a set of sentences true in w_0 and L is a set of relevant laws/generalizations.

Of course, there are singular facts of the evaluation world (on top of the fact $\neg A$) that will entail the antecedent to be false, no matter how the notion of entailment relevant for (5) is defined. The reason is that the fact that the antecedent is false in the evaluation world is not a fact that stands completely on its own. If, for instance, we use a causal notion of entailment, then the direct or indirect causes of the antecedent will entail the antecedent to be false in the evaluation world. This causes a serious thread to the scheme (5). One could counter that this only asks for a very careful spell-out of the scheme: the definition of the set $B(w_0)$ has to make sure that there are no sentences in this set that entail that the antecedent is false given $\models$. But it seems not very plausible that such a specification of $B(w_0)$ is possible. There appear to be facts of w_0 that we need to be in $B(w_0)$ in order to account for the truth conditions of counterfactuals, but still they entail that the antecedent is false.

Consider, for instance, the following example involving causal reasoning. Plants need both, sun and water to grow; they will die if one of the two is missing. Furthermore, lets assume that we live in a part of our planet where sun means a lot of sun and therefore drought. We are aware of this problem and therefore installed a sprinkler system that is automatically switched on when there is sun. With s for *there is sun*, d for *the plants die*) and p for *the sprinkler is on* this scenario can be captured by the causal laws $s \leftrightarrow p$ and $s \wedge p \leftrightarrow \neg d$). Lets also assume that the actual world is such that the sun is up and burning (s), the sprinkler is on (p) and the plants are doing just fine ($\neg d$). Now consider the counterfactual conditional (6) in this context.

(6) If the sprinkler system had be off, the plants would have died.

This counterfactual is intuitively true in the described context. But this means that in the counterfactual scenario considered by the antecedent of (6) still the sun has to be burning. From this together with the laws it follows that the plants will die. Thus, in order to account for (6) we have to assume that s an element of $B(w_0)$. But s also (causally) entails p in the considered situation. Hence, we have to make sure that while s needs to be an element of $B(w_0)$, $B(w_0)$ together with the antecedent $\neg p$ does not entail p.

This is basically the same kind of non-monotonic reasoning pattern described by (4), with A_1 corresponding to the causal law $s \leftrightarrow p$, A_2 relating to the fact s that the sun is shining - both facts of the evaluation world, and B corresponds to $\neg p$, the antecedent of the counterfactual. So, we see that there are at least two different ways in which conditionals can give rise to non-monotonic inferences: first, explicitly mentioning exceptional circumstances can defeat established laws/ regularities, and second, we can notice directly a violation of a given law/regularity when observing that its antecedent is clearly true, while still its consequent does not hold. The goal of the present paper is to account within a uniform framework for both types of non-monotonic reasoning patterns, independent of whether we are concerned with reasoning with conditionals or reasoning within conditionals.[2]

3. Minimal models and counterfactual conditionals

The non-monotonicity of conditionals is standardly approached using the Minimal Models framework. This line of approach originates in the work of Stalnaker and Lewis, not only the idea to use minimal models to describe the meaning of conditional sentences, but also the idea of minimal models itself as well a an investigation into their logical properties. The basic idea is to define a strict or partial order $\leq_M$ on the class of models for your formal language and relative to a selected model M. Intuitively, this order compares models with respect to their similarity to M. Given this order, we can specify the set $Min(A, \leq_M)$ of the models making A true that are minimal with respect to the order. Then one can define a new notion of entailment as given in (7). This notion of entailment is non-monotonic, because it only takes a subset of the models making the premises true into account. Extra premises can lead to a completely different set of relevant models.

(7) $A \models_{\leq_M} C$ iff all $\leq_M$-minimal models of A also are models of C, i.e. in case $Min(A, \leq_M) \models C$.

This notion of entailment is then used to spell out the scheme (1). This results in the definition of truth conditions for conditionals paraphrased in (8).

[2] We do not want to claim that the classification of non-monotonic inference patterns for conditional sentences given here is complete. Indeed, we will discuss in the conclusions shortly another kind of non-monotonicity conditionals can give raise to.

(8) A counterfactual with antecedent A and consequent C is true in a model M in case all models of A that are most similar to M also are models of C.

Depending on the definition of the order this approach can model all kinds of non-monotonicity effects. The approach is extremely powerful and flexible. There are no restrictions on the type of model and nearly no restrictions on the type of order. You can bend this approach any way you need and want it. Another plus is that it is also very intuitive and simple to understand. Therefore, it is not surprising that it is at the moment the by far most used approach to non-monotonic reasoning in linguistic and philosophy, certainly when it comes to conditional sentences.

However, it has also a couple of disadvantages. A central problem is that depending on the order and the type of models you work with minimal models can be very hard to access. Therefore, it can be difficult to prove general results for the application we are interested in or calculate predictions for concrete examples. This is related to another problem: reasoning with minimal models is highly complex, and therefore, it has been argued, not cognitive plausible.[3]

For these reasons it is interesting to look for a formal framework for non-monotonic reasoning that is less complex and easier to work with. The particular alternative framework we want to have a look at is *logic programming*.

4. Approaching conditionals with Logic Programming

Lets have a look on how we can model the non-monotonic behavior of conditionals using logic programming. We will do so in three steps. We start with the standard system of logic programming where negation is not allowed. As we will see, this system is not appropriate for our application. In a next step we will discuss the system of (Stennings and van Lambalgen, 2005) which will turn out to be able to deal with some, but not all of the inferences we want to model. Finally we will have a look at a system that combines (Stennings and van Lambalgen, 2005) with (Schulz, 2011) and will show that in this system we are able to account for both of the non-monotonic inference patterns discussed in section 2.[4]

[3] See (Stennings and van Lambalgen, 2005).

[4] It should be pointed out that in this paper we will take a semantic perspective on logic programming. We do so because in the last section we want to discuss the relation between approaching counterfactual conditionals using logic programming

4.1. The standard approach

The general idea behind the logic programming approach to non-monotonic reasoning is very similar to the approach using minimal models. Also in this case only a subgroup of the models that make the premisses true are relevant for the reasoning. However, these particular models – actually it is one singular model – are now selected in a different way.

We can distinguish three steps that take us to this model and define a particular system of logic programming. First, we have to define the formal language in which the given information (the premisses) are expressed. The language is a fragment of propositional (or predicate) logic. Sentences of this language are called program clauses, and a set of such sentences, representing the given information, is called a program. Definition 1 describes the relevant formal language for the standard approach.

DEFINITION 1. *Let $\mathcal{L}$ be a set of proposition letters. A (positive) clause is a formula of the form $p_1, ..., p_n \rightarrow q$, where $p_1, p_2, ..., q$ are propositional variables; the antecedent may be empty. In this formula, q is called the head, and $p_1, ..., p_n$ the body of the clause. A (positive) program is a finite set of positive clauses.*

Basically, one can interpret a program as encoding two types of information. On the one hand, there is law-like information encoded in the conditional clauses with non-empty antecedents. They describe dependencies between the proposition letters. On the other hand, there are the clauses with empty antecedents. They are normally taken to encode singular facts.

The next ingredient we need is a class of models for the programs. The standard approach works with simple two-valued interpretations M ($M : \mathcal{L} \longrightarrow \{0, 1\}$). Truth for sentences with respect to a model M is defined according to classical logic. We write $M \models \phi$ in case the sentence ϕ is true in model M. We also define an order on the models comparing point-wise the interpretation of the proposition letters. For two models M_1, M_2 we define $M_1 \leq M_2$ iff $\forall p \in \mathcal{P} : M_1(p) \leq^+ M_2(p)$, where $\leq^+$ is defined on $\{0, 1\}$ as the relation $\{\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 1\rangle\}$.

The final step is to select for a given program a particular model. This model is then used to define a non-monotonic notion of entailment (something follows from the program if it is true in this particular model). We select the model by defining – using the program – a function on the class of models. This function (called an immediate

and using minimal models. This relation can be discussed more easily taking also a semantic approach to logic programming.

consequence operator) intuitively computes the immediate predictions of the laws (as specified in the program) given the facts of the model.

DEFINITION 2. *The operator $\tau_{P,1}$ associated with a program P is a function on the set models. For every proposition letter $p \in \mathcal{L}$ the value $\tau_{P,1}(M)(p)$ is defined as follows.*

1. $\tau_{P,1}(M)(p) = 1$ *if there is a program clause in P with p as head, and the body of this program clause is true in M.*

2. $\tau_{P,1}(M)(p) = 0$ *otherwise.*

The function $\tau_{P,1}$ turns out to be monotone, independent of the choice of program. That means that for all models M_1, M_2 it holds that if $M_1 \leq M_2$, then $\tau_{P,1}(M_1) \leq \tau_{P,1}(M_2)$. Such functions are known to have fixed points, i.e. there are models M such that $\tau_{P,1}(M) = M$. The least fixed point is the point we use as the selected model on which we will check what follows from the information encoded in the program P.

DEFINITION 3. *For a positive program P let $\tau^*_{P,1}$ be the smallest fixed point of a operator τ_P. Then we define for any $\phi \in \mathcal{L}$: $P \models^+ \phi$ iff ϕ is true in $\tau^*_{P,1}$.*

Assume, for instance, we are given the information that p is true and that r depends on p and q to be true. This results in the positive program $P = \{\rightarrow p, p \wedge q \rightarrow r\}$, i.e. we have two program clauses, the first with an empty body. Now, we can ask ourselves what follows from this information given the notion of entailment we have just defined. The least fixed point is the model $M = \tau^*_{P,1}$ with $M(p) = 1$, $M(q) = 0$ and $M(r) = 0$. It is easy to see that the clauses in P are all true in $\tau^*_{P,1}$. Thus $\tau^*_{P,1}$ is a model of P. But it is a very specific model. We also have $P \models^+ \neg q$ and $P \models^+ \neg r$. More generally, every proposition letter for which it doesn't follow from the facts and laws of P that it is true is set to false. Thus, $\models^+$ models negation as failure.

This notion of entailment is too harsh for the particular application we are interested in. Thinking back to the first type of non-monotonic reasoning displayed by conditionals, reasoning with or within conditionals does not apply negation as failure all over the board. Only abnormality conditions, exceptions to laws/generalizations should be negated if possible. Thus, while it is nice that we have a reasoning mechanism that can account for negation as failure, we only want it applied to a particular subset of the propositional variables. So, what

we need to add is a way to mark a propositional variable in a way that can be recognized by the immediate consequent operator.

Another problem with this approach is that the program language is too restrictive for our purposes. In order to be able to express rules with abnormality conditions/exceptions we need to allow for negation in the body of program clauses (we need the abnormality conditions to occur under negation in the body of program clauses). The approach presented in the next subsection will deal with both of these problems.

4.2. The approach of Stenning & van Lambalgen

The next version of logic programming we are going to discuss has been explicitly designed to deal with reasoning with abnormality conditions introduced in section 2 (see (Stennings and van Lambalgen, 2005)). The first difference with the previous system is that it allows for negation in the body of program clause. But in order to accommodate this change in expressibility we need to switch to a different class models for programs. If we allow for negation in the body of program clauses and keep the old two-valued notion of model, then the immediate consequence operator τ (in its straight forward extension to deal with negative bodies) will no longer be monotone, and, in consequence, does not need to have fixed points anymore.[5]

To deal with this problem (Stennings and van Lambalgen, 2005) propose to switch to a different logic for programs. It distinguishes truth values $\{u, 0, 1\}$ with the partial order $u \leq 0$ and $u \leq 1$. The chosen ordering reflects the intuition that u can 'evolve' towards one of the values 0 and 1. The three-valued truth tables for the connectives $\neg$, $\wedge$ and $\vee$ are given in figure 1.[6] Definition 4 extends the order on truth values to models.

DEFINITION 4. *For two three-valued models M_1, M_2 we define $M_1 \leq M_2$ iff $\forall p \in \mathcal{P} : M_1(p) \leq M_2(p)$.*

This switch to a different class of models doesn't make the immediate consequence operator monotone again. We also need to adapt the operator to the extra truth value just introduced. Definition 5 does that.

[5] A simple program that illustrates this point is the program consisting of the two clauses $\neg p \rightarrow q$ and $q \rightarrow p$.

[6] For the definition of the immediate consequence operator it is sufficient to fix the connectives $\neg$, $\wedge$ and $\vee$ (actuallly, we don't need $\vee$ either). Thus, at this point we don't need to decide on the truth table of the implication.

p	0	1	u
$\neg p$	1	0	u

p	0	0	0	1	1	1	u	u	u
q	0	1	u	0	1	u	0	1	u
$p \wedge q$	0	0	0	0	1	u	0	u	u

p	0	0	0	1	1	1	u	u	u
q	0	1	u	0	1	u	0	1	u
$p \vee q$	0	1	u	1	1	1	u	1	u

Figure 1. Three-valued truth tables for $\neg$, $\wedge$ and $\vee$

DEFINITION 5. *The operator $\tau_{P,2}$ associated to a program P is a function on the set models. For every propositionletter $p \in \mathcal{P}$ the value $\tau_{P,2}(M)(p)$ is defined as follows.*

1. $\tau_{P,2}(M)(p) = 1$ *if there is a clause $\phi \rightarrow p$ in P such that $M \models \phi$.*
2. $\tau_{P,2}(M)(p) = 0$ *if there is a clause $\phi \rightarrow p$ in P and for all such clauses $M \models \neg\phi$.*
3. $\tau_{P,2}(M)(p) = u$, *otherwise.*

The function $\tau_{P,2}$ turns out to be monotone. So, we can be certain that a least fixed point exists[7] and define the notion of entailment analogously to definition 3. As shown in (Hölldobler and Ramli, 2009) for any program P the least fixed point is identical to the least model of the weak completion of P under Łucasiewicz semantics.[8,9]

Except for the introduction of the third truth value, there is an additional difference between $\tau_{P,2}$ and $\tau_{P,1}$ that is essential for the inferences that can be derived based on these two operations. The operation $\tau_{P,1}$ implicitly assumes that we have a completely picture of the positive facts - thus, if in the database there was no information to the point that some propositional variable was true, then it had to be false. The new operation $\tau_{P,2}$ implicitly assumes that we have a complete picture of the possible "causes" of some propositional variable to be true. In case a model makes all propositional variables on which some variable

[7] It can be reached by iterating $\tau_{P,2}$ starting with the empty model, see (Hölldobler and Ramli, 2009).

[8] To get the weak completion of a program one first replaces the set of all clauses with the same head $Body_1 \rightarrow A$, $Body_2 \rightarrow A$, ... by the sentence $(Body_1 \vee Body_2 \vee ...) \rightarrow A$, and then exchanges all occurrences of $\leftarrow$ with $\leftrightarrow$.

[9] Łucasiewicz semantics means that we have the following truth table for the implication and bi-implication:

p	1	1	1	0	0	0	u	u	u
q	1	0	u	1	0	u	1	0	u
$p \rightarrow q$	1	0	u	1	1	1	1	u	1
$p \leftrightarrow q$	1	0	u	0	1	u	u	u	1

.

p depends false, the immediate consequence operator concludes that also p is false.

But can this system deal with the problem that for abnormality conditions we still want the old version of negation as failure: as long as there is no information to the contrary, the system should derive that nothing abnormal is going on? Indeed, it can: by writing for abnormality predicates a dummy reason for falsity in the program: we allow for program clauses with $\bot$ as body.

DEFINITION 6. *Let $\mathcal{L}$ be a set of proposition letters. A (definite) clause is a formula of the form $(\neg)p_1 \wedge ... \wedge (\neg)p_n \rightarrow q$, where $p_1, ..., p_n$ are either propositional variables, $\top$ or $\bot$, and q is a propositional variable. Empty antecedents are not allowed. A definite logic program is a finite conjunction of definite clauses.*

Given this definition of programs, abnormality conditions are marked: these are the heads of clauses with $\bot$ as body. And without extra information these variables will come out as false. But if we have information to the contrary, in fact it is sufficient to know that an abnormality condition depends on some other variable(s), then negation as failure will no longer be applied.

This also shows that clauses of the form $\bot \rightarrow p$ cannot be interpreted as negative facts. Only in case there is no additional information in the program the head of these clauses comes out as false in the minimal fixed point. There is no way to express negative facts in these programs. As before, program clauses with non-trivial body represent dependencies, program clauses with $\top$ as body can be understood as positive facts, and now additionally, we have clauses with $\bot$ as body, which represent abnormality conditions.

Lets also illustrate this extended approach with an example. We take the same program as discussed in the last section: $P = \{\top \rightarrow p, p \wedge q \rightarrow r\}$. Now, the body of the "fact" p is not empty, but $\top$. The least fixed point $\tau^*_{P,2}$ is the model that takes p to be true and q and r to be undefined. As we see, negation as failure is applied to neither of these two variables. Both are not marked as abnormality condition and, hence, also not treated as one. If we add the clause $\bot \rightarrow q$, then the least fixed point will take both, q and r to zero. The variable q because it now is marked as an abnormality condition and there is no information contradicting that q is false, the variable r because all the conditions on which this variable depends (there is only one: $p \wedge q$) are known to be false.

The approach of (Stennings and van Lambalgen, 2005) can very nicely

account for examples like (3).[10] One might complain that it is strange that we need to tread abnormality predicates differently in the program. But every approach will need to distinguish them from other variables that represent standard singular facts, simply because they behave differently. In this case the marking of an abnormality predicate is directly linked to the modeling of their distinguishing behavior, which is rather elegant.

The approach can also account for examples like (2) concerning non-monotonicity within conditionals. Just to sketch how would work: we say that an conditional with antecedent A and consequent C is true in case the program P entails C ($\tau^*_{P,2} \models C$), where P contains next to $\top \rightarrow A$ all laws/dependencies the speaker considers in the utterance contexts.[11] For example (2) we could assume the following generalizations $P = \{p \wedge \neg ab \rightarrow q, \bot \rightarrow ab\}$ with p for *I strike this match*, q for *the match is lit* and ab for *something abnormal is going on*. If we add to this set the antecedent of the conditional (2-a) ($\top \rightarrow p$), the resulting program will entail the consequent of the conditional. But if we extend the generalizations with the clause $t \rightarrow ab$ with t for the match his wet, then we can't derive anymore that the match will lit from the antecedent of (2-b).[12]

When discussing example (4) we said that we also want to be able to model the fact that observing the head of a program clause to fail leads to the conclusion that we are facing a law violation. The approach discussed here cannot account for this observation for two reasons. First of all the approach does not allow for negative facts (i.e. negation in the head of a program clause) and therefore we cannot model the observation (fact) that the head of some program clause does not hold. This is a technical problem of the logic programming framework we work with. The other reason is that the way the immediate consequence

[10] For a detailed discussion see (Stennings and van Lambalgen, 2005).

[11] We need to add some additional tools in order to deal with logically complex antecedents.

[12] Notice the difference between the program clause $p \wedge \neg ab \rightarrow q$ and a conditional with antecedent p and consequent q in this case. The second is a linguistic expression for which we want to establish a truth value. The first is a law we assume to be valid in the context in which the conditional is evaluated. This approach has a certain circular feeling to it because it might look as if we explain the truth of a conditional based on the conditional itself. But that is misleading. The (causal) dependencies connecting an antecedent p to a consequent q can be much more complex and indirect, meaning that we need a different and more complex program to establish the truth of the conditional. It is just that there can be in this case a direct correspondence between a perceived dependency and the conditional that is evaluated - and we assume that for this example this is the case. Of course this approach raises questions. For instance, nothing is said about how to determine the correct program to evaluate the truth conditions of a certain conditional.

operator is defined laws (program clauses) cannot be cancelled. Every application of the operator $\tau_{P,2}$ will always try to fix the value of the heads in the output model to match the value of the bodies in the input model. Thus, even if the notion of program would allow for negation in the head of program clauses and we would be able to model the scenario in (4), we would still not predict the desired outcome, because the operator $\tau_{P,2}$ would override the observation that the wineglass didn't break in order to maintain the validity of the law that if a glass is dropped, it will break. The goal of the next subsection will be to overcome this shortcoming.

But before we continue with this final proposal, we should shortly address another way to account for the pattern exemplified by (4), which keeps the present version of the approach unchanged. One could argue that when encountering a violation of a law the reasoning mechanism should not simply deal with the problem by allowing for an exception, but rather it should lead to a reconsideration of the program. Apparently, the laws formalized in the program don't describe reality correctly. Probably some abnormality condition is in play that has not been considered before. So, one might want to argue that the approach of (Stennings and van Lambalgen, 2005) already describes non-monotonic reasoning with conditionals completely. We just need on top of it some additional mechanism adapting programs when facing law violations. However, so far we don't have such an extension of the theory of (Stennings and van Lambalgen, 2005). It is also not clear that this is the right strategy to approach the problem. While it looks a plausible option to account for (4), it does not seem to capture correctly what is going on in (6). Also on a more conceptual level it is questionable that every time we face a law violation we stop drawing inferences and first reset the program. For these reasons we will account for (4) without involving revision of the program.

4.3. A final approach

We already outlined at the end other previous subjection what needs to be done. In order to account for the reasoning pattern exemplified in (4) we need (i) to allow for negative facts, and (ii) to adapt the definition of the immediate consequence operator in a way that facts take precedence over laws: facing a fact that stands in conflict with a law, we assume an exception to the law and not rewrite the facts guided by the laws.

In order to deal with the first problem we no longer take facts to be part of the program, but model them by selecting a particular starting model to which the immediate consequence operator is applied. Let Σ

be a set of literals, thus Σ may contain positive and negative facts. We define M_Σ to be the model for which it holds that for every $p \in P$ if neither p nor $\neg p$ are elements of B, then $M_\Sigma(p) = u$; if $\neg p \in B$, then $M_\Sigma(p) = 0$, and if $p \in B$ then $M_\Sigma(p) = 1$. M_Σ is the minimal three-valued model that makes the given facts true. The definition of programs remains unchanged, as does the definition of the models and the order over models, but now programs are taken to only represent the given dependencies, and no singular facts.[13]

We solve the second problem with a new definition of the immediate consequence operator. The new operator only changes the value of a propositionletter if it is so far undefined (i.e. has value u). Everything that at some point is determined as either true or false stays this way.

DEFINITION 7.
The operator τ_P associated to a program P is a function on the set models. For every propositionletter $p \in \mathcal{P}$ the value $\tau_P(M)(p)$ is defined as follows.

(i) *If $M(p) \neq u$ then $\tau_P(M)(p) = M(p)$.*

(ii) *If $M(p) = u$, then*

a. *$\tau_P(M)(p) = 1$ if there is a clause $\phi \rightarrow p$ in P such that $M \models \phi$.*

b. *$\tau_P(M)(p) = 0$ if there is a clause $\phi \rightarrow p$ in P and for all such clauses $M \models \neg\phi$.*

Normally, we would now proceed with using the least fixed point of this operator τ to be the model with respect to which we define our non-monotonic notion of entailment. Unfortunately, there is a problem. The operator τ is not monotone.[14] This means that we cannot simply rely on fact **??** to derive the existence of fixed points for τ_P.

However, we have something else for this operator, which turns out to be sufficient for the existence of fixed points: for all programs P and all models M it holds $M \leq \tau_P(M)$. This follows immediately from the definition of τ. But then, at least if we work with a finite vocabulary, the iterative application of τ to a model M converges after finitely many steps. The limit, $\tau_P^*(M)$, is a fixed point of τ_P. This fixed point can then serve, again, as the selective model we use to define a notion of entailment (definition 8).

[13] It is still possible to have $\top$ as the body of a program clause. We can use such clauses to express necessary truths.

[14] Take, for instance, the model M_1 that maps p to 1 and q to u and the model M_2 that maps p to 1 and q to 0. Our program contains only one clause: $p \rightarrow q$. It is clear that $M_1 \leq M_2$, but one can also easily see that $\tau(M_1) \not\leq \tau(M_2)$.

DEFINITION 8. *A set of literals Σ entails a sentence ϕ given a program P, $\Sigma \models_P \phi$, in case ϕ is true on the fixed point $\tau_P^*(M_\Sigma)$ reached after iteratively applying τ to M_Σ.*

This approach can deal with the problematic inference (4). Let p stand for the proposition *you drop a wineglass* and q for *the wineglass breaks*. Then we can formalize the first two premisses as $p \wedge \neg ab \rightarrow q$ (for A_1) and p (for A_2). The program describing this scenario is $P = \{p \wedge \neg ab \rightarrow q, \bot \rightarrow ab\}$. Lets start with checking whether C follows from A_1 and A_2. This translates in this framework into the question whether $\Sigma \models_P q$ with $\Sigma = \{p\}$. To answer this question we need to apply τ_P iteratively to the model M_Σ, which assigns the value u to every proposition letter (q and ab) except p and assigns to p the value 1. The table below states the calculation of the iterative applications of τ_P until the fixed point $\tau_P^*(M)$ is reached. As one can see, in the fixed point nothing abnormal is going on ($\tau_P^*(M)(ab) = 0$) and the glass breaks ($\tau_P^*(M)(q) = 1$). In other words, we get the intended result $\Sigma \models_P q$.

model	p	ab	q	
M_Σ	1	u	u	
$\tau_P(M_\Sigma)$	1	0	u	
$\tau_P(\tau_P(M_\Sigma))$	1	0	1	
$\tau_P(\tau_P(\tau_P(M_\Sigma)))$	1	0	1	$\longleftarrow$ fixed point $\tau_P^*(M_\Sigma)$

If we now add the fact that the glass didn't break, i.e. work with $\Sigma' = \{p, \neg q\}$, we start with a different initial model (the model now assigns to q the value 0) and end up with a different fixed point (see the table below). In this fixed point still nothing abnormal is going on, but the glass stays broken. In other words, the selected model does not make the program true. This is a distinctive feature of the present approach.

model	p	ab	q	
$M_{\Sigma'}$	1	u	0	
$\tau_P(M_{\Sigma'})$	1	0	0	
$\tau_P(\tau_P(M_{\Sigma'}))$	1	0	0	$\longleftarrow$ fixed point $\tau_P^*(M_{\Sigma'})$

So far we have seen that the new approach can deal with the non-monotonic reasoning pattern exemplified by (4) that (Stennings and van Lambalgen, 2005) cannot capture. Though we applied the approach here to an example involving reasoning with conditionals, it can also be used to account for similar observations concerning reasoning within

conditionals (example (6)).[15] But as an extension of (Stennings and van Lambalgen, 2005) it can also deal with the pattern of non-monotonic reasoning that we illustrated using (2) and (3).[16] Thus, we conclude that the present approach improves on (Stennings and van Lambalgen, 2005) in terms of descriptive power.

5. Relating Logic Programming to minimal models.

The goal of this last section is to study the relation between approaching the non-monotonicity of conditionals using minimal models with the approaches relying on logic programming discussed in section 4. For semanticists and philosophers, who normally work within the minimal models framework, the first impression from reading section 4 might be that the Minimal Models approach is simpler. There is no need for many valued logic, no immediate consequence operator for which we have to prove the existence of fixed points etc. However, one shouldn't forget that this simplicity is directly related to the generality of the approach. If you want a minimal models approach to make concrete predictions, you need to add details, for instance, define a concrete notion of model and an order over these models.

It has been argued, and not by the least (e.g. Stalnaker), that we should adopt a minimal models approach to conditional sentences exactly because of its generality. The semantics of conditionals is so flexible and context dependent that we cannot say more about their semantics than what is stated in the interpretation rule (8) on page 7. But, as we have seen in the second section of this paper, there are certain patterns in the the non-monotonicity of conditionals. A rule like (8) does not capture these patterns. For this we would need to specify particular orders over models. This shows that in its simple general form the minimality approach is not adequate.

The Logic Programming framework clearly wins when it comes to computational complexity and usability. It is easy to apply the approach to examples and calculate predictions. Maybe the most pressing disadvantage of this line of approach is that even in the form developed in subsection 4.3 there are still substantial restrictions on the expressibility. We cannot have arbitrary conditional sentences as program clauses, and the set of premises to which the new notion of entailment is applied needs to be a set of literals. This has not been a problem for

[15] The latter is discussed in detail in (Schulz, 2011), though there a simpler system is used that does not take into account abnormality conditions.

[16] For these examples the predictions are exactly the same as made by (Stennings and van Lambalgen, 2005).

the examples discussed here, but that might easily change in the future with other examples. However, until these limitations in expressibility turn out to constitute a real problem, there a good reasons to prefer the logic programming approach over a minimal models approach to conditional sentences.

Still, one might wonder what exactly the relation is between approaching conditionals with minimal models or logic programming. Do they truly differ? Is there a way to define the same inference relation introduced in definition 8 also working with minimal models? In other words, given a program P is there an order $\leq_P$ such that the following holds?

(9) $\quad \Sigma \models_{\leq_P} \phi$ iff $\Sigma \models_P \phi$

In one sense you might say that the answer is trivial. We can view the approach presented in the last section as specifying a selection function for optimal models. This function takes a program P and a set of literals Σ and returns the selected model $\tau^*_P(\Sigma)$. We can easily define an order based on $\tau^*_P(\Sigma)$ that returns the set of worlds[17] that are the completions[18] of the three valued model $\tau^*_P(\Sigma)$. We just compare worlds with respect to their similarity to the selected model $\tau^*_P(\Sigma)$. Then, we say that Σ entails ϕ in case ϕ is true on the minimal worlds with respect to this order. This notion of entailment would be equivalent to $\models_P$. But what we have sketched here is in a strict sense not a minimal models approach, because the order depends on the premise set Σ. What we want is an order that just depends on P and still gives us the equivalence (9).

There is a well-known result about the relation of orders and selection functions that we might want to use: a selection function f can be translated into a strict weak order that selects the same worlds a minimal, if it has the following properties (the domain of the function f is the set O of all finite subsets of the set of possible worlds):

(C) $\forall p \in O : f(p) \neq \emptyset$.
(A1) If $p \subseteq p'$, then $p \cap f(p') \subseteq f(p)$.
(A2) If $p \subseteq p'$ and $p \cap f(p') \neq \emptyset$, then $f(p) \subseteq f(p')$.

Unfortunately, we cannot use this result, because $\tau^*_P(\Sigma)$ is not a selection function in this sense: it is only defined for sets of literals.

[17] A (possible) world is a three-valued model that assigns either 1 or 0 to each proposition letter.

[18] A completion of a three-valued model is a possible world that agrees with the model on the valuation of all proposition letters that the model evaluates as either 1 or 0.

We could try to overcome this limitation and extend the approach to arbitrary sets of premises Σ. This is not discussed here, because it is very hard to say what the predictions should be for premise sets containing disjunctions. Assume, for instance, the program contains $p \rightarrow q$. What should be entailed by a premise set $\Sigma = \{p \vee \neg q\}$ for this program? We leave this issue for future research. But notice that how this problem is solved will have consequences for the equivalence (9). For instance, (9) can only hold in case $\tau^*_P(\Sigma)$ makes the same predictions for logical equivalent sets of premises.[19]

Thus, we have to accept that the standard route from selection functions to orders is presently not an option. But maybe we can still define an order that gives us (9); one that works at least for those cases that the logic programming approach can deal with, i.e. for premise sets consisting of literals. In the reminder of this section I will argue that for the special case of acyclic programs, we cannot establish (9), but we can get pretty close.

Lets think a moment about what kind of order we need to establish (9). What the order $\leq_P$ basically has to do is to minimize law violations: a possible world should deviate al least as possible from what the program P predicts. But not all law violations count the same. Suppose, for instance, the program is $P = \{A \rightarrow B, B \rightarrow C\}$ and the observations, i.e. the premise set, is $\{A, \neg C\}$. We have two possible worlds that both model the premise set and have only one law violation: w_1 maps A and B to 1 and C to 0, and w_2 maps A to 1 and B and C to 0. The logic programming approach introduced in section 4.3 goes for world w_1. More generally, this approach allows for law violations only at the latest possible moment (in w_1 the law $B \rightarrow C$ is violated, while in w_2 it is the law $A \rightarrow B$). For (9) to go through the order $\leq_P$ needs to work the same way.

Lets sketch how we could define an order that does this. Assume we have an acyclic program P. Then, this program can be associated with a tree structure O^*_P: we let O_P be the tuple $\langle \mathcal{L}, < \rangle$, such that for every $p, q \in P$ it holds $p < q$ if there is a program clause $p_1, ..., p_n \rightarrow q$ in P and p is among $p_1, ..., p_n$, and define O^*_P as the transitive closure over O_P. We can now assign numbers to the proposition letters measuring how far away from the root, the minimal elements of the structure O^*_P, a proposition letter is.[20] Lets call this number the *height* of a proposition

[19] This would, for instance, mean that given the program $P = \{p \rightarrow q\}$ the premise $(p \wedge q) \vee (p \wedge \neg q)$ needs to entail q, because p entails q.

[20] Given that we work with a finite vocabulary and acyclic programs, there will be minima for each proposition letter.

letter.[21] Now, we define for each height n an order $\leq_n$ that compares possible worlds with respect to the law violations for program clauses with a head of height n. Based on these orders we define the order $\leq_P$ as follows:

$$\begin{array}{l} \text{For all } w_1, w_2 \in W\text{:} \\ \quad w_1 \leq_P w_2 \quad \text{iff}_{\text{def}} \quad w_1 \leq_0 w_2, \text{ and if } w_1 =_0 w_2 \text{ then} \\ \qquad\qquad\qquad\quad\; w_1 \leq_1 w_2, \text{ and if } w_1 =_1 w_2 \text{ then} \\ \qquad\qquad\qquad\quad\; \ldots \\ \qquad\qquad\qquad\quad\; w_1 \leq_n w_2. \end{array}$$

With this order $\models_{\leq_P}$ will entail everything that $\models_P$ entails. But it will entail actually more. It predicts backtracking in certain circumstances, i.e. it can reason from the head to the body of a program clause.[22] To illustrate, assume the program is $P = \{A \to B, B \to C\}$, and $\Sigma = \{B\}$. Then we get as $\tau_P^* \Sigma$ the model that maps A to u, and B and C to 1. Consequently, $\Sigma \not\models_P A$. But the possible world the order $\leq_P$ picks out as optimal maps all three proposition letters to 1, and hence $\Sigma \models_{\leq_P} A$.

There is no way to prevent this effect without making the order sensible to the set of premises Σ. Thus, the conjecture (9) fails to hold. This shows that there are differences between the logic programming approach and using minimal models beyond those we have already discussed: they can give rise to different predictions. This gives us another reason to chose one of the approaches above the other. But deciding between them now becomes an empirical question, one that lies beyond the scope of this paper.

We finish with two remarks on the differences in predictions the two approaches make. First of all, there has been discussion about backtracking in the context of conditional sentences. (Lewis, 1979), among others, has argued that counterfactual conditionals do not allow for backtracking, but the empirical situation is far from clear. However, "backtracking" as discussed in the literature on conditionals does not necessarily refer to the very specific case of backtracking the order $\leq_P$ predicts.[23] Thus, taking a particular stance towards backtrack-

[21] Again, given that we work with a finite vocabulary and acyclic programs, there will be a maximal height. We will use this fact in the definition of the order.

[22] In a causal interpretation of the program clauses we can describe this kind of reasoning as reasoning from effect to cause: if you observe a certain effect, you can conclude that there was a cause.

[23] The minimal models approach predicts backtracking only for variables that are in the sense of O_P^* smaller than or completely unrelated to the variables in Σ. For all other variables no backtracking is predicted. We can easily define a semantics for

ing for counterfactuals does not immediately decide between the two approaches discussed here.

There is another, related observation about differences between the two approaches. It concerns abnormality predicates. Logic programming predicts that based on a program $P = \{p \wedge \neg ab \rightarrow q, \bot \rightarrow ab\}$ we derive from observations $\Sigma = \{p, \neg q\}$ the sentence $\neg ab$. One could argue that in case we observe that a law fails we reason that something abnormal is going on. So, we should predict ab. It is easy to get this prediction just with a very small variation in the definition of the order sketched above. For logic programming, because of its inherent dynamics, this is much harder.[24] Thus, if this prediction is what we want, a minimal models approach to reasoning with laws does somewhat better than logic programming.

We leave it with this. As will have become clear, there are still many open questions that need to be addressed in future work. But the answers involve for a substantial part empirical work on the particular application we have discussed here. How the details of the logic programming approach have to be spelled out, or whether we should prefer logic programming over minimal models depends on what exactly we want to predict for reasoning with or within conditionals.

This goes beyond what this paper set out to achieve. The main objective here was to establish the attractiveness of logic programming as a tool to model non-monotonic reasoning. We have tried to establish this by focussing on one particular application where non-monotonic reasoning plays an important role: conditional sentences. As we have seen, one can capture characteristic properties of the non-monotonistic behavior of conditionals very nicely with logic programming. Maybe the strongest feature of using logic programming here, compared to a minimal models approach, is that the specific reasoning patterns we set out to model come natural to logic programming, while, as we have seen in this last section, one has to use rather complex orders to achieve the same effects with a minimal models approach. But, as said above, from a descriptive point of view there is still a lot to be done before we can say which line of approach is more adequate for conditional sentences.

---

conditionals using the order $\leq_P$ where such variables do not occur (we would use the notion of basis introduced in (Schulz, 2011).

[24] There exists some proposals for abductive reasoning with logic programming that could be used, for instance (Kakas et al., 1993).

References

Byrne, R.: 1989, 'Suppressing valid inferences with conditionals'. *Cognition* **31**, 61–83.

Goodman, N.: 1955, *Fact, fiction and forecast*. Indianapolis/New York/Kansas City: The Bobbs-Merrill Company, Inc.

Hölldobler, S. and C. K. Ramli: 2009, 'Lodic programs under three-valued Łukasiewicz's semantics'. In: H. P. M. and D. S. Warren (eds.): *Logic programming*, Vol. 5649. Heidelberg: Springer, pp. 464–478.

Kakas, A., R. Kowalski, and F. Toni: 1993, 'Abductive Logic Programming'. *Journal of Logic and Computation* **2**(6), 719–770.

Lewis, D.: 1979, 'Counterfactual dependence and time's arrow'. *NOÛS* **13**, 455–476.

Schulz, K.: 2011, 'If you wiggle A, then B will change. Causality and counterfactual conditionals'. *Synthese* **179**, 239–251.

Stennings, K. and M. van Lambalgen: 2005, 'A working memory model of relations-between interpretation and reasoning'. *Cognitive Science*.

Veltman, F.: 2005, 'Making counterfactual assumption'. *Journal of Semantics* **22**, 159–180.