Counterfactuals: A Fresh Look at Non-Monotonic Reasoning

Trevor Bench-Capon and William Hounslow
Department of Computer Science
University of Liverpool
PO Box 147
Liverpool
L69 3BX
UK
Tel: +51-794-3697
Fax: +51-794-3715
E-mail: tbc@compsci.liverpool.ac.uk

21 December, 1993

Abstract

This paper considers the problem of non-monotonic reasoning and suggests an alternative line of attack to the current, excessively formal, methods. The approach attempts to explicate the nature of defaults used in systems of non-monotonic logic by comparing them to counterfactual conditionals, a topic of long standing interest in philosophy. It takes a particular approach to counterfactuals, which treats them as elliptical arguments, and shows how this can form the basis of a computationally effective treatment. This is illustrated by the presentation of a treatment of a model based diagnosis problem using this approach.

Content Areas: common sense reasoning, diagnosis, non-monotonic reasoning, counterfactuals

Declaration: This paper is unpublished, original work, substantially different from papers currently under review and will not be submitted elsewhere before the notification date other than to workshops and similar specialized presentations with a very limited audience.

1 Introduction

Default knowledge - which seems to be the cornerstone of attempts to capture common sense knowledge - typically takes the form of statements such as

(1) Birds can fly.

or

(2) Basketball players are tall.

Note that these expressions are not quantified: although both (1) and (2) command our assent, the corresponding quantified statements

(3) All birds can fly

and

(4) All basketball players are tall

are less well founded. (3) is clearly false, since it is well known that there are many types of birds incapable of flight, and whilst (4) may currently be a true universal generalisation, it is at least possible that a short basketball player should appear on the scene.

Once we have represented the default knowledge in a system, however, it will be applied to all objects of the appropriate type - effectively we are using (3) and (4) - which we have already agreed are either false, or at least need to be handled with care. Including these dubious statements in a theory will inevitably give rise to problems, because they will license deductions that we do not wish to be made. This in turn means that we need a way to distinguish the conclusions we want to draw on the basis of a default rule from those we wish to block. Informally the most active strategy is to draw conclusions from default rules, but only as long as it is consistent so to do. The problem then arises that we may draw a conclusion which is consistent, but then acquire a new piece of information which renders it inconsistent. If this happens we need a mechanism to enable us to withdraw the conclusion.

In other words reasoning with default knowledge appears to be non-monotonic in character: whereas in classical logic a correctly drawn deduction from a theory cannot be overturned by any subsequent extension to the theory, it appears that common sense is different.

This is the problem of non-monotonic reasoning. Now, because deduction is a pervasive model of reasoning, the temptation has been to question not the statements themselves, but the logic used to manipulate them. In so far as the statements have been questioned, the standard approach is to qualify them, in the manner of the *ceteris paribus* rider traditional among economists, which has become the 'ab' predicate when used in computer science.

But this is no more than an admission of ignorance in the case of economists, and a purely technical solution in the case of computer scientists. Common sense does not, in fact, assent to either

(5) Birds, other things being equal, can fly,

or to

(6) Birds, unless they are abnormal, can fly.

Neither of these are acceptable, because a kiwi is not an abnormal bird simply because it is flightless, nor is being a kiwi sufficiently strange to permit us to say that the *ceteris paribus* clause has been violated. True, no one would cite a kiwi as a typical bird, but there are quite a number of species of flightless bird, and people are quite familiar with kiwis, ostrichs and penguins, and have no difficulty in recognising them as birds. If we are refining the notion of bird sufficiently to permit consideration of individual species, it is part of the normal run of things to take flightless birds into account. The *ceteris paribus* could be used to rule out wholly exceptional circumstances, such as the unfortunate bird having its feet set in concrete, but not a common occurrence such as being a member of a fairly familiar species of bird.

The computational complexity of approaches based on detecting inconsistency are well documented. See, for example, [6]. The complexity issues, are important. Not only do they impede the practical realisation of these theories, but they also cast doubt on their power to explain common sense reasoning. How can common sense reasoning necessitate computations of such a nature? It has been often argued that the transitive closure of their beliefs cannot be attributed to people: how much more doubtful is the detection of inconsistencies amongst such a transitive closure which additionally includes the use of 'default' rules?

The truth about a misapplication of (1) is that it is not the bird that is abnormal, but the circumstances in which the rule is being used. The circumstances assumed by non-monotonic logic, require both transitive closure of beliefs, and a mechanism for detecting and resolving inconsistencies. This cannot be common sense. Most people would simply say that it is simply rather rash to use (1) to draw a conclusion, if the situation is such that it is relevant to consider the species of bird.

2 Counterfactual Conditionals

The work on non-monotonic logics has drawn its inspiration largely from work on formal logic. Given a calculus of timelessly true propositions, it might seem natural to try to develop a similarly motivated calculus for default statements. The work is of undoubted interest, but seems to us to move away from rather than towards a satisfying explanation of common sense reasoning. The lack of real progress in terms of practically applicable results should be enough to encourage us to try a new track. For this reason we might consider what insights we can derive from philosophical, as opposed to formal, logic.

In philosophical logic there has been a long standing interest in a particular, strange, form of the conditional, termed the subjunctive, or counterfactual, conditional.

A counterfactual is a conditional in which the antecedent and the consequent are in the subjunctive mood.

(7) If I were a bird I would be able to fly

or

(8) If I were a basketball player I would be tall.

Such conditionals are strange because they lack three properties that are true of normal conditionals.

- They are non-monotonic: strengthening the antecedents may cause the counterfactual to become unacceptable. Thus

  (9) If I were a bird and a kiwi I would be able to fly

  is not acceptable, even though (7) is.

- They are non-transitive. Whereas for normal conditionals

  ```
  P -> Q
  Q -> R
  ------
  P -> R
  ```

  is a valid argument, the same is not true of counterfactuals. Consider

  (10) If I lived in Eire I would be a Catholic

  (11) If I were a Catholic I would go to the Liverpool Metropolitan Cathedral.

  Both of these may be true, since I but

  (12) If I lived in Eire I would go to the Liverpool Metropolitan Cathedral.

  is not, since, admirable as the Cathedral is, it is hardly worth a ferry trip to attend Mass each Sunday.

- Contraposition fails for counterfactuals. Thus for normal conditionals the argument:

  ```
  P -> Q
  -Q
  ------
  -P
  ```

  is valid, the same move cannot be made with counterfactuals. Thus from (7) and

  (13) I am unable to fly

it cannot be concluded that I am not a bird, since I could be a bumble bee or a bluebottle.

The arresting thing is that all three of these properties of counterfactuals are true of defaults as well. We must decide whether there are two kinds of strange conditionals, counterfactuals and defaults, which share all their important properties, and one of which has been overlooked throughout philosophical history, or whether they are in fact one and the same. And if the latter, does it not make sense to look carefully at philosophical theories of counterfactuals, to see if they give us some computational leverage? An immense amount of effort has been devoted to the exploration and formalisation of non-monotonic logics: our belief is that the effort would have been better applied to the correct explication of counterfactual conditionals.

3 Treatments of Counterfactuals

Two major theories of counterfactuals emerge from philosophy: one, deriving from the work of Stalnaker [10] and Lewis [7], explains them in terms of closest possible worlds, the other, deriving from the work of Mackie [8], contends that they represent elliptical arguments. Unfortunately it is the solutions in terms of possible worlds that seem to have received the most attention by those in AI who have looked into counterfactuals. See for example [4]. We say this is unfortunate because we believe it to be both philosophically wrong, and computationally unhelpful. It is computationally unhelpful because it also depends on detecting inconsistencies in possible worlds, so that the possible world closest to the actual world in which the antecedent holds can be determined and examined to see whether the consequent also holds. This has unfavourable complexity implications, as has been discussed by Eiter and Gottlob [3]. The philosophical problems turn on finding a satisfactory way of giving content to the notion of distance between possible worlds.

Mackie's account treats counterfactuals, not as statements akin to logical sentences, but as *elliptical arguments*, having all but one premise suppressed. A counterfactual is deemed to be acceptable if some set of premises can be produced that enable the connection between antecedent and consequent to be demonstrated. To assert a counterfactual, then, is to imply that one could, on request, produce the suppressed premises, and to deny it is either:

- to deny one of the suppressed premises;
- to deny that the consequent follows from the antecedent plus the suppressed premises;
- to provide another counterfactual with the same antecedent and a denial of a suppressed premise as a consequent.

3.1 Arguments Are Made in a Context

A key advantage of speaking in terms of arguments is that the notion brings with it the idea of *context*. Context is crucial in two ways: first, an argument is typically presented against a background of shared information, so that it is rare indeed for all the premises of an argument to made explicit. Second, the context determines the acceptability of the premises, and determines the extent to which statements must be qualified. Suppose we are constructing an argument in which the boiling point of water is required as a premise. For most purposes

(14) Water boils at 100 degrees centigrade

will serve, even though the boiling point of water varies with pressure and the purity of the water. These qualifications need to be made to (14), however, only if the context is such that it is necessary to consider the effects of impurities or pressure. Moreover if the context is boiling water in the Alps, we need only make the qualification with regard to pressure, and if the context is boiling water in Portsmouth (where the water contains a lot of chalk), we need only consider the effect of impurities. When discussing arguments, the acceptability of, and the degree of qualification that must be made to, premises is determined by the context, the theory within which they will be used.

Applying this lesson to default knowledge we can see that whether a statement such as (1) is acceptable can only be seen from considering the context of the theory within which it will be used, and the problem which that theory is designed to solve. If no mention is made of species of bird in the rest of the theory, (1) is likely to be acceptable, but if the theory mentions kiwis, ostrichs and penguins, we need to qualify it to

(15) Birds, other than penguins, kiwis and ostrichs, can fly

Note that we need only mention those species that appear elsewhere in the theory: we need not mention cassowaries, since there is no way of expressing that something is a cassowary in the theory. Non-monotonicity arises when we extend the theory. Certain extensions, in particular those which introduce new concepts to the theory, may change the context so as to invalidate certain statements in the theory, by forcing us to qualify them further. The importance of context in common sense problem solving is discussed at length in [5].

3.2 Representing Counterfactual Conditionals

If we construe counterfactuals as elliptical arguments, we can get a simple, computationally effective treatment, provided we restrict ourselves to horn clauses for modelling the domain, since in such cases we need not worry about contradictions because we have no way of positively establishing negations. This solution was originally proposed and justified in Bench-Capon (1989), which used Mackie's approach to formalise as part of logic program counterfactual statements occurring in law. This makes full use of the context sensitivity of arguments, and hence counterfactuals: the context is identified with the theory represented by the knowledge base, and the acceptability of a counterfactual is determined by the ability to prove the consequent from the union of the knowledge base and the antecedent. Note that we intend to use goal driven reasoning to execute our theory: this means that the theory is not regarded as a possible world - which needs to be consistent - but rather as a pool of premises which can be used as needed to fill out the elliptical argument of the counterfactual.

To enable us to represent counterfactuals in the body of a clause we define a predicate `counterfactually(A,C)` where `A` is a list of antecedents and `C` is a list of consequents. `counterfactually(A,C)` is thus to be read *If all A were true, all C would be true.*

```
counterfactually(A,C):-assert-all(A),call-all(A,C),retract-all(A).

call-all(A,[]).
call-all(A,[H|T]):-call(H),call-all(A,T).
call-all(A,L):-retract-all(A),fail.

assert-all([]).
assert-all([H|T]):-asserta(H),assert-all(T).

retract-all([]).
retract-all([H|T]):-retract(H),retract-all(T).
```

The straight representation of counterfactuals is even simpler: we can simply write the counterfactual as a horn clause, relying on the fact that it will be interpreted within a closed theory to give it its acceptability.

Given this we can represent counterfactuals explicitly in our knowledge base. We will now attempt to show some of the advantages of this capacity by showing how they are used in a solution of the problem of diagnosing faults in the full adder. We have chosen this problem because it forms the subject of Reiter's well known paper [9], in which he demonstrated the use of his characterisation of non-monotonic reasoning as applied to model based diagnosis.

4 Full Adder Problem

The circuit of interest is shown diagramatically in Figure 1. It is a one bit adder with carry in and carry out. There are three inputs and two outputs: one output, the sum line, is on if an odd number of inputs are on, and the other, the carry line, is on if at least two inputs are on.

[width=3.5in]fulladder.eps

Figure 1: Circuit for Full Adder

The strategy to diagose faults is to construct a model of a working circuit, and then supply a set of inputs and outputs. If this data fails to conform to the predictions of the model, we then use the model to determine which components are such that they satisfy the counterfactual

> If this component were faulty it would give rise to this set of observations.

This will, of course, require us to state what would happen were particular components faulty.

4.1 Modelling A Working Circuit

4.1.1 Modelling the Gates

First it is necessary to model the functions of the gates of various types.

```
out(X,Y):-andg(X), working(X), !,
          evaluatea(X,Y).

evaluatea(X,t):- in(X,1,t), in(X,2,t).
evaluatea(X,f):- in(X,1,f).
evaluatea(X,f):- in(X,2,f).

out(X,Y):-org(X), working(X),!,
            evaluateo(X,Y).

evaluateo(X,f):- in(X,1,f), in(X,2,f).
evaluateo(X,t):- in(X,1,t).
evaluateo(X,t):- in(X,2,t).

out(X,Y):-xorg(X), working(X),!,
              evaluatex(X,Y).

evaluatex(X,f):- in(X,1,f), in(X,2,f).
evaluatex(X,f):- in(X,1,t), in(X,2,t).
evaluatex(X,t):- in(X,1,t), in(X,2,f).
evaluatex(X,t):- in(X,1,f), in(X,2,t).

out(X,Y):-input(X), working(X),!,
          in(X,Y).
```

Two points should be are worthy of brief comment First we use cuts in the definitions of `out`: this is to declare that a gate can only have a single output. Second we have a definition of `out` for inputs: this enables the modelling of the connections between the components to be done in a uniform fashion, whether the signal is an input to the circuit or an output from a gate.

Finally we state that a component is working if it is not faulty:

```
working(X):- not faulty(X).
```

Because we will be using negation as failure, this means that a component will be presumed to be working if there is no explicit information as to its status. This use of negation as failure does not, however, take us out of standard logic: whenever a goal is run it is done so in the context of the theory, and in that context the theory is, in line with our discussion of arguments above, taken to be complete. This means that, at the time the query is made, our procedures can be interpreted as individually sufficient and jointly necessary, and so we can use the notion of 'completion' of the database [2] to align the behaviour with that of standard logic.

4.1.2 Connections

In Reiter's original model [9] the role of connections was implicit, and so his model is committed to the assumption that all the connections are working correctly. To give ourselves the ability to relax this assumption we model connections explicitly.

```
in(Y,N,T):-connected(X,Y,N,C), working(C),!,
          out(X,T).
```

where connected(X,Y,N,C) is to be read as C is the connection between the output of X and the Nth input of Y. Represented in this way the possibility of the connection being faulty will arise: if we wish to make the assumption that no connections are faulty we can exclude this possibility by removing the condition that the connection be working:

```
in(Y,N,T):-connected(X,Y,N,C),
          out(X,T).
```

Essentially this makes the assumption that all the connections are working by removing the possibility of expressing the contrary. This is a manifestation of our belief that qualifications are context dependent rather than absolute: the qualification that the connection is working is made only in a context where the faultiness of connections *needs* to be considered.

4.1.3 Components

We now declare the set of components in our circuit

```
andg(a1).
andg(a2).
org(o1).
xorg(x1).
xorg(x2).
input(i1).
input(i2).
input(i3).
```

```
connected(x1,a2,2,c1).
connected(x1,x2,1,c2).
connected(a2,o1,1,c3).
connected(a1,o1,2,c4).
connected(i1,x1,1,c5).
connected(i1,a1,1,c6).
connected(i2,x1,2,c7).
connected(i2,a1,2,c8).
connected(i3,a2,1,c9).
connected(i3,x2,2,ca).
```

For convenience we also summarise the components in our circuit:

```
components([x1,x2,a1,a2,o1,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca]).

gates([x1,x2,a1,a2,o1]).
```

4.1.4 Finding the Outputs

We can now write a simple procedure to find the outputs of all the gates given a set of inputs.

```
dogates(Y):-gates(G),calcouts(G,[],Y).

calcouts([],Y,Y).
calcouts([H|T],S,Y):-out(H,X),calcouts(T,[[H,X]|S],Y).
```

`dogates` takes the list of gates and calculates the output for each of them in the context of the current inputs.

We now have a model of the circuit which give the outputs of a working circuit. For example, if we supply the inputs used in [4],

```
in(i1,t).
in(i2,f).
in(i3,f).
```

`dogates(X)` will return:

```
X = [[o1,f],[a2,f],[a1,f],[x2,t],[x1,t]]
```

4.2 Introducing Faults

4.2.1 Faulty Components

Given this model of a working circuit, we now need to be able to say what would happen were the circuit faulty. First we need to say what would happen if a component were faulty. The intuitive answer is that if a gate is faulty it gives as output the opposite of what it would give were it working. Note that we have to use the subjunctive mood here: we do not wish to say that the faulty gate will always give the opposite of the working gate, since the gate may for example be faulty because it always signals on or off. The use of the counterfactual, however, allows us to place the statement in the context of a set of observations: in this situation the faultiness of the gate affects the observations, and so it must be giving the incorrect output. Were the output fortuitously correct, the output of the circuit as a whole would be correct, and no explanation of unusual behaviour would be needed.

We thus use an explicit counterfactual to represent the required behaviour:

```
out(X,Y):-faulty(X),counterfactually([working(X)],[out(X,Z)]),
          switch(Z,Y).

switch(t,f).
switch(f,t).
```

We can apply the same reasoning to the connections

```
in(Y,N,T):-connected(X,Y,N,C), faulty(C),
           out(X,U), switch(U,T).
```

Here, however, we do not need a counterfactual in the body of the clause, since the faultiness of the connection does not affect the signal it *should* transmit, which is determined by the gate from which it receives the signal. Thus we have no need to determine what would be the case were a faulty gate working. The clause as a whole does, however, represent the counterfactual:

> if the gate were faulty, its output would be the opposite of its input.

Alternatively we could reason that if the connection were faulty its output would always be off:

```
in(Y,N,f):-connected(X,Y,N,C), faulty(C).
```

We can use which of the definitions of a faulty connection we please, and obtain the results corresponding to the desired interpretation.

4.2.2 Diagnosing Faults

The diagnosis problem arises when we have some observations which do not conform to our expectations. These observations can be simply represented as a list, such as:

```
observations([out(x2,f),out(o1,f)]).
```

Note that the output of the gate `x2` is incorrect for the inputs suggested above. We can now state what it is for a gate to be faulty in the context of these observations. Again the most natural formulation is in the subjunctive mood:

> A component X is possibly faulty if and only if: if X were faulty it would explain all the observations.

We therefore make further use of our `counterfactually` predicate:

```
all-faults([],O,X,X).
all-faults([H|T],O,S,X):-counterfactually([faulty(H)],O),
                         all-faults(T,O,[H|S],X).
all-faults([H|T],O,S,X):-all-faults(T,O,S,X).
```

Here we process a list of components, represented by the first argument: if the faultiness of the component would explain the observations in the second argument, it is added to the list of such components in the third argument. When all the components have been tested, the complete list of such components is returned in the fourth argument. We can now write a predicate which will identify all the possible faults for a given set of components and observations:

```
possible-faults(X):-components(C),
                    observations(O),
                    all-faults(C,O,[],X).
```

With these definitions we are now able to identify possible faults. Thus, for example, given the inputs and observations described above, issuing the query `possible-faults(X)` would result in the answer:

```
X = [x2,x1].
```

Indicating that it must be one of the xor gates that is faulty. Similarly, if we relax the assumption that the connections are working, by including `working` in our definition of `in` and including in our theory the definition that says that a faulty connection always outputs false, we would get:

```
X = [c5,c2,x2,x1].
```

indicating that the fault might lie in the first xor gate, the connection between that gate and its input which is on, the second xor gate, or the connection between the two xor gates.

This will pinpoint the suspect components, but it is of no help in suggesting to the user further tests to discriminate between two candidate faults. We therefore added an interface which prints a table of possible faults, and the outputs that *would* result from each of them. Here again we require the use of an explicit counterfactual.

The resulting output, using the above inputs and assuming no faulty connections, is:

```
   o1  a2  a1  x2  x1
x1 f   f   f   f   f
x2 f   f   f   f   t
```

From this we can see that we need to inspect the output of the first xor gate in order to see which of the xor gates is faulty.

If we allow connections to be faulty and use the definition whereby faulty connections give the opposite of what they should give we get:

	o1	a2	a1	x2	x1
x1	f	f	f	f	f
x2	f	f	f	f	t
c2	f	f	f	f	t
c5	f	f	f	f	f
c7	f	f	f	f	f
ca	f	f	f	f	t

If, on the other hand we assume that all faulty connections output false we get:

	o1	a2	a1	x2	x1
x1	f	f	f	f	f
x2	f	f	f	f	t
c2	f	f	f	f	t
c5	f	f	f	f	f

4.2.3 Multiple Faults

In the above there is an implicit assumption that only one component is faulty. This assumption is made in `possible-faults`, where the list of components passed to all faults ensures that components are examined one at a time. It is, however, a simple matter to look instead at pairs of components. We can change `possible faults` so that the predicate reads

```
possible-faults(X):-components(C),
                    setof(D,combs(C,2,D),E),
                    observations(O),
                    all-faults(E,O,[],X).
```

The predicate `combs(List,N,ListofN)` unifies `ListofN` to a selection of `N` items from `List`. Now all pairs of components will be tested to see if they explain the observations. Running this on the data used by Reiter [9], so that the inputs are on, off and on and the outputs are on and off we get, making the assumption that connections are not faulty:

	o1	a2	a1	x2	x1
[x2,a2]	f	f	f	t	t
[x2,o1]	f	t	f	t	t

showing the two possible pairs of faulty gates, and that testing the output of the second and gate will discriminate between them. If desired, we could make a similar extension to combinations of any specific number of faults. If we wanted to find an indefinite number of faults we would need to leave the second argument of `combs` as a variable, and write a predicate which would call it successively instantiated to 1,2,3 etc.

This is not the place to discuss the full complexity implications of our approach. However, the key source of the problems that arise in the other approaches - the need to detect inconsistencies - does not apply to our method. Our theory is executed in standard Prolog using the standard Prolog execution strategy. This has the critical advantage of focussing the search towards the goal we are trying to establish. Moreover, the goals using `counterfactually` are treated like any other goal. Thus, if complexity issues do arise they do so for reasons unconnected with our method of representing defaults as counterfactuals.

5 Conclusion

Much attention - we would say too much attention - has been paid to the development of non-monotonic logics to enable reasoning with so-called default information. The result has been increasingly formal accounts, which, whilst interesting in their own right, are unsatisfying as explanations of common sense reasoning. We therefore turned to philosophical accounts of counterfactuals - conditional statements which share three important properties with expressions of default knowledge - for guidance on how to represent default knowledge. Our favoured interpretation of counterfactuals treats them as elliptical arguments. If we construe expressions of default knowledge as deriving from such context sensitive counterfactual knowledge, we can both sensibly decide whether we wish to include such information, and what qualifications we need to place upon it in the context and problem framework we are working within. Moreover, if we model our domain in horn clauses we have a computationally effective method of exploiting such knowledge. The example implementation of a model based diagnosis problem given above is intended to lend support to these views.

References

[1] Bench-Capon, T.J.M. (1989). 'Representing Counterfactual Conditionals', in *Proc. Seventh Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour*, ed. A. G. Cohn. London: Pitman.

[2] Clark, K.L., (1983). 'Negation As Failure' in *Logic and Databases'* H.Galaire and J.Minker (eds). New York: Plenum Press.

[3] Eiter, T., and Gottlob, G. (1992). 'On the Complexity of Propositional Knowledge Base Revision, Updates and Counterfactuals', *Artificial Intelligence*, *57*, pp. 227–270.

[4] Ginsberg, M.L. (1985). 'Counterfactuals', *Artificial Intelligence*, *30*, pp. 35–79.

[5] Hounslow, W.E., ‘A Possible Worlds Approach to Common Sense Reasoning’, PhD Thesis, The University of Liverpool, 1993.

[6] Kautz, H.A., and Selman, B., ‘Hard Problems for Simple Default Logics’, *Artificial Intelligence*, *49*, pp243-79.

[7] Lewis, David (1973). *Counterfactuals.* Oxford: Basil Blackwell.

[8] Mackie, J. L. (1962). ‘Counterfactuals and Causal Laws’, in *Analytical Philosophy*, ed. R. J. Butler. Oxford: Basil Blackwell.

[9] Reiter, R. (1987). ‘A Theory of Diagnosis From First Principles’, *Artificial Intelligence*, *32* , pp. 57-95.

[10] Stalnaker, R. (1968). ‘A Theory of Conditionals’, In N. Rescher(ed) *Studies in Logical Theory.* Oxford: Blackwell.