Consistency Constraints for Input/Output Logic: A Comparative Review

David Makinson and Leendert van der Torre

Abstract

In a range of contexts, one comes across processes resembling inference, but where input propositions are not in general included among outputs, and the operation is not in any way reversible. Examples arise in contexts of conditional obligations, goals, ideals, preferences, actions, and beliefs. In a separate paper, we developed a general theory of such processes when they are applied without restriction. In this paper, we compare systematically several ways of restricting them by consistency constraints.

1. Background

We assume familiarity with (Makinson and van der Torre, this volume), which defines and studies unrestricted output operations. Nevertheless, for convenience, we briefly recall the central points.

We work in a boolean context, that is, we consider a propositional language closed under the usual truth-functional connectives. The central objects of attention are ordered pairs (a,x) of formulae, which we read forwards, i.e. with a as body and x as head. Intuitively, we think of the body a as representing a possible *input*, and the head x as a possible *output*. We call a set G of such pairs a *generating set*. The letter G also serves as a reminder of the interpretation (among others) of the pairs as conditional goals or obligations. When A is a set of formulae, we write $G(A)$ for $\{x\colon (a,x) \in G$ for some $a \in A\}$.

The operation $out(G,A)$ takes as argument a generating set G, and an input set A of formulae, delivering as value a set of formulae. On the semantic level:

- *Simple-minded output*, written $out_1(G,A)$, is defined as $Cn(G(Cn(A)))$;
- *Basic output*, written $out_2(G,A)$, is defined as $\cap\{Cn(G(V))\colon A \subseteq V,\ V$ complete$\}$, where a *complete* set is one that either is maxiconsistent or is equal to the set L of all formulae of the language;
- *Simple-minded reusable output*, written $out_3(G,A)$, is defined as $\cap\{Cn(G(X))\colon A \subseteq X = Cn(X) \supseteq G(X)\}$;
- *Basic reusable output*, written $out_4(G,A)$, is defined as $\cap\{Cn(G(V))\colon A \subseteq V \supseteq G(V),\ V$ complete$\}$.

We have $out_i(G,A) \subseteq Cn(A\cup m(G))$ for each $i = 1,2,3,4$, but not in general conversely. Here $m(G)$ is the set of all materialisations of elements of G, i.e. the set of all formulae $b\rightarrow y$ with $(b,y) \in G$.

On the syntactic level, we work with singleton inputs, with derivability from an input set A understood as derivability from the conjunction a of finitely many elements of

A. Moreover, in the syntactic context it is more convenient to work with the notation '$(a,x) \in out(G)$' than with '$x \in out(G,a)$'. We thus consider rules for deriving pairs (a,x) from a generating set *G*, which is also a set of pairs.

In general, for any set of rules, we say that a pair (a,x) of formulae is *derivable from G using those rules* iff (a,x) is in the least set that includes *G*, contains the pair (t,t) where *t* is a tautology, and is closed under the rules. The specific rules considered are:

SI:	From (a,x) to (b,x) whenever $b+ a$
AND:	From (a,x), (a,y) to $(a,x\wedge y)$
WO:	From (a,x) to (a,y) whenever $x+ y$
OR:	From (a,x), (b,x) to $(a\vee b,x)$
CT:	From (a,x), $(a\wedge x,y)$ to (a,y).

As shown in the companion paper, simple-minded output coincides with derivability using SI, AND, WO; basic output to those plus OR; simple-minded reusable output to the first three plus RT; and basic reusable output to all five.

Except where explicitly indicated to the contrary, the term 'output' will be used to cover indifferently all four output operations, and we write simply $out(G,a)$ instead of '$out_i(G,a)$ for $i = 1,2,3,4$'. To lighten language, we sometimes refer to basic reusable output simply as *reusable output*.

All of our input/output operations satisfy replacement of input, and of output, by classically equivalent propositions. That is, if $(a,x) \in out(G)$ then $(a',x') \in out(G)$ whenever $Cn(a) = Cn(a')$ and $Cn(x) = \mathrm{Cn}(x')$. It will be convenient to treat replacement of logically equivalent propositions as a 'silent rule', that may be applied at any step without explicit justification.

2. Motivation and Questions

In this paper we consider what happens when consistency constraints are imposed on the output operation. The motivation for imposing such constraints comes from the logic of norms (deontic logic), where they have been proposed as one important element involved in making good sense of 'contrary-to-duty' conditional obligations.

Let *G* be a code consisting of explicitly given conditional obligations, and let *a* be a condition. If the code tells us that *a* ought not to be the case, then it is customary to say that any conditional norm (a,x) with *a* as body is 'contrary-to-duty'. It is widely accepted that when assuming such a condition *a*, the norms of the code forbidding it $out_i(G,A) \subseteq Cn(A\cup m(G))$ become in some sense inoperative. We recall a well-known example.

Example 1. 'Reykjavik scenario' (Belzer 1987). Consider a code with three elements, the first saying that neither Ronald (Reagan) nor Mikhael (Gorbatchov) should be told the secret, the second that if Ronald is told then so should Mikhael, and the third that if Mikhael is told then so should Ronald. Schematically, the code is $G = \{(t,\neg r\wedge\neg m), (r,m), (m,r)\}$. Now consider the condition *r* that Ronald is in fact told, which is contrary-to-duty since (t,¬r) is implied by the code. It is natural to conclude, under that condition, that Mikhael should also be told. Moreover, under the same condition, it appears incorrect to conclude the opposite, that Mikhael should *not* be told, despite the fact that this is part of the content of the first element of the code. It would also seem pointless and perhaps incorrect to conclude, under the same condition, its own opposite - i.e. that Ronald should not be told. Schematically, (r,m) should be derivable, but (r,¬m) and (r,¬r) should not. But the unrestricted operations put all three in *out*(G), the two undesired ones by simple applications of SI and WO.

We do not pretend to have a recipe for dealing with examples such as these. The literature already contains a wide variety of approaches. When the principles of the code are understood as defeasible, then it is natural to use some kind of prioritisation of its elements, notably in terms of degree of specificity. However, it is debatable whether prioritisation is consistent with a strictly indefeasible reading of the elements of the code, that is, as never in any circumstances admitting any exceptions. There are also examples, such as the following, where prioritisation does not appear to help anyway.

Example 2. Möbius strip (Makinson 1999). Put $G = \{(a,b), (b,c), (c,\neg a)\}$, where the pairs are understood as conditional goals, intentions or obligations. All of b, c, ¬a are in the reusable output of G under input a. Intuitively, we would be inclined to accept b as output, and also c if we admit reusability, but loath to accept ¬a.

In this example, there does not seem to be any straightforward way of prioritising the elements of G in terms of degree of specificity,. There are three maximal subsets H of G whose unrestricted (reusable) output with input a is consistent with that input, namely the three two-element subsets. Intersecting the outputs for all of them gives $Cn(\emptyset)$, and the only choice among the three values of H that gives the 'right' result is $H = \{(a,b), (b,c)\}$.

On the other hand, intuition is easily accommodated in the example by using derivation rules, with consistency checks at the nodes of the derivation trees. One simply 'keeps going' as long as possible until hitting trouble, where 'trouble' means violation of a consistency constraint. In other words, in any derivation, at any node n with attached pair (a,x), the input a considered at n should be consistent with the entire collection of pairs attached to nodes on which n depends. Consider the simplest derivation of (a,¬a):

$$\dfrac{\dfrac{(a,b) \qquad \dfrac{(b,c)}{(a\wedge b,c)}\,\text{SI}}{(a,c)}\,\text{CT} \qquad \dfrac{(c,\neg a)}{(a\wedge c,\neg a)}\,\text{SI}}{(a,\neg a)}\,\text{CT}$$

This is blocked at the last step, since the input *a* considered there is inconsistent, in some sense, with the pairs on which it depends, in particular with the three leaves.

But what kind of consistency is this? It is not just classical consistency, for it compares a proposition with one or more *ordered pairs* of propositions And how exactly should dependence be defined? The first purpose of this paper is thus:

- *To articulate ways in which these notions may be unpacked*. We shall see that there are at least four ways of reading consistency, and two of dependence; and that some combinations sit together more comfortably than others (sections 3,4).

Our further purposes are:

- *To compare the force of consistency constraints under the various options*. In general they are distinct, although some are stronger than others. For derivations not using OR, the two notions of dependence trivially coincide; one of our main results (Observation 3) is that for such derivations, the four notions of consistency also agree.

- *To study the role of leaves and the root node in the application of consistency constraints*. We have some positive and some negative results for both derivations and derivability (Observations 7-9,11-13).

- *To seek semantic characterisations of constrained output.* The positive results on the role of leaves and roots, permit us to characterise restricted output semantically when the unrestricted version does not use OR (Observation 10). For output using OR, the problem is not settled.

We emphasise that we are not advocating a particular solution to the problem of interpreting contrary-to-duty obligations. Our intention is formal. It is to articulate and compare, more systematically and carefully than has been done before, some main kinds of consistency constraint on derivations, that may eventually form part of such a solution.

3. Concepts of consistency

Consider any derivation Δ and a node $n{:}(a,x)$ of Δ. Let H be the set of nodes of Δ on which n depends (in a sense later to be made precise). In the context in which we are working, perhaps the most principled way of understanding the consistency of the body a with H is in terms of output itself. In other words, in any derivation, when $n{:}(a,x)$ is a node depending on H, we should require:

Body/output (bo) constraint wrt H: $\neg a \notin out(H,a)$.

Another way of understanding the consistency of a with H, is in terms of the materialisation $m(H)$ of H , i.e. (section 1) the set of all formulae $b \rightarrow y$ with $(b,y) \in G$.

Body/materialisation (bm) constraint wrt H: $\neg a \notin Cn(m(H))$.

This, in turn, has two close neighbours. One requires the body a to be consistent with the set $h(H)$ of all *heads* of elements of H; the other with the set $f(H)$ of all *fulfilments* of elements of H, where the fulfilment of (b,y) is defined as $b \wedge y$. In other words:

> *Body/head (bh) constraint wrt H*: $\neg a \notin Cn(h(H))$
> *Body/fulfilment (bf) constraint wrt H*: $\neg a \notin Cn(f(H))$.

The *bf* constraint has been examined intensively by van der Torre in a number of publications, notably (van der Torre 1997, 1998, 1999), of which the last provides a succinct overview. The *bh* constraint has been studied in the context of the logic of conditional norms in (Makinson 1999). The *bm* constraint does not appear to have received much attention in the literature. As far as the authors are aware, the *bo* constraint has not been investigated or even isolated. We shall consider all four in a comparative manner.

Observation 1. For fixed a and set H of pairs, $bf \Rightarrow bh \Rightarrow bm \Rightarrow bo$ where the arrow means that satisfaction of the left constraint (at a with respect to H) implies satisfaction of the right one.

Proof. Clearly, the materialisation $b \rightarrow y$ of a node (b,y) is implied by its head y, which in turn is implied by its fulfilment $b \wedge y$; this shows the first two implications. For the third, recall from section 1 that $x \in out_i(H,a)$ implies $x \in Cn(\{a\} \cup m(H))$. Putting $x = \neg a$ and applying classical logic we thus have in particular that $\neg a \in out(H,a)$ implies $\neg a \in Cn(m(H))$.

However, as we shall now see, the appropriate notion of dependence (i.e. the value of H chosen when applying the constraint to a derivation) may take different forms in derivations using the rule OR, rendering the relationship between the four constraints more complex. Roughly speaking, for *bo* and *bm* we should consider the entire subtree generated by the node under consideration, whereas for *bh* and *bf* we should split it at applications of the OR rule.

4. Concepts of dependence

On what nodes in a derivation tree would we say that a given node $n:(a,x)$ 'depends'? The natural first answer is: those occurring in the subtree determined by n, travelling leafwards. Evidently, one may query whether n itself may (or should) be excluded from the subtree. We will show later that if n is not a leaf, then the decision makes no difference. But there is another one that does.

As observed in (Makinson 1999), for derivations with the rule OR, taking the entire subtree as the dependency set can give inappropriate results under the *bh* and *bf* constraints, since it bars proof by cases except when the cases are mutually consistent.

Example 3. Proof by exclusive cases (Makinson 1999): Put $G = \{(a,x \wedge y), (\neg a,x \wedge \neg y)\}$, and consider the following derivation to get (t,x) where t is a tautology.

```
(a,x∧y)              (¬a,x∧¬y)
   |    WO                  |        WO
(a,x)                  (¬a,x)
---------------------------------   OR
                 (t,x)
```

Intuitively, the derivation appears acceptable when the pairs are understood as conditional goals or obligations. But formally, the situation is as follows. The first steps, using WO, pass all four checks, since on the left $a \wedge x \wedge y$ is consistent and similarly on the right so is $\neg a \wedge x \wedge \neg y$. For the last step, let H be the set of nodes of the tree. On the one hand, its materialisation $m(H)$ is consistent, so $\neg t \notin Cn(m(H))$. Thus the last step derivation passes the *bm* check (and so also the *bo* check). On the other hand, the set of all heads of elements of H (indeed of the leaves in H) is inconsistent. Thus $\neg t \in Cn(h(H)) \subseteq Cn(f(H))$ and the derivation fails the *bh* and *bf* checks.

For this reason, when investigating respectively the *bh* and *bf* constraints, (Makinson 1999) and (van der Torre 1998, 1999) work with a more discriminating account of dependence, with a special treatment of branching under the OR rule. Roughly speaking, it is thought of as creating parallel dependency tracks. When we apply it to pass from premises (a,x) and (b,x) to $(a \vee b,x)$, the latter may be seen as depending on each of the two premise nodes considered separately, but not on them taken together.

In general, a node depends on each member of a family of sets of nodes, as follows. Given a node n in a derivation tree, we travel leafwards in the subtree generated by n, splitting it into two at every application of the OR rule, so that one tree contains one of the two premise nodes and the other tree contains the other premise node. In this way we obtain a family of trees, each of which is part of the subtree generated by n. We say that node n depends *separately* on the set of nodes of *each* member of that family. This definition is rather informal, but is visually clear and suffices for our purposes without entering into the formalities of graph theory.

Example 4. Dependency sets. Consider a derivation tree with rules applied as follows (we give names to the nodes but abstract from the formulae):

```
A       B                C                D                E
---------- CT            |                | SI             |
    F                    |                G                |
     ----------------------- OR           ------------------ OR
                  H                                 I
                   ----------------------------------- AND
                                    J
```

Then the root node J depends on each of the following four sets: {J,H,F,A,B,I,G,D}, {J,H,C,I,G,D}, {J,H,F,A,B,I,E}, {J,H,C,I,E}. With unsplit dependence, it depends on the set of all nodes of the tree. As remarked above, one might prefer to exclude J itself from these sets and, as will be seen later, it makes no difference to the resulting consistency constraints.

In this light, we return to the weaker constraints *bo* and *bm*, and ask whether it would make any difference if we split trees there too. It would, as shown by the following example.

Example 5. Effect of splitting on bo, bm. Put $G = \{(a,x), (\neg a,x)\}$, and consider the following derivation.

$$\begin{tikzcd}
(a,x) \arrow[d, dash, "\text{WO}"] & & (\neg a,x) \arrow[d, dash, "\text{WO}"] \\
(a,x\vee y) \arrow[dr, dash] & & (\neg a,x\vee y) \arrow[dl, dash, "\text{OR}"'] \\
& (t,x\vee y) \arrow[d, dash, "\text{SI}"] & \\
& (\neg x, x\vee y) &
\end{tikzcd}$$

If, in our definition of dependence, we split this tree at the OR step, then it passes the *bm* (and hence also *bo*) consistency check. For example, looking at the body $\neg x$ of the root node, it is consistent with the set of materialisations of nodes in the left branch, and likewise with those of the right branch. But if we do not split the tree, then the derivation fails the *bo* check (and thus also *bm*) at its root, since $x \in out_2(H,\neg x)$ where H is the set of all nodes (indeed, leaves) of the tree.

Intuitively, the authors feel that split dependence seems to accord with the idea of demanding consistency of the current body with heads and fulfilments of previously used items, whilst unsplit dependence accords with requiring consistency with their materialisations or output. From here on, when we speak of the various constraints, we shall always understand them with dependence notions associated in that way unless expressly indicated to the contrary.

In other words, the four consistency constraints that we shall consider are the following, for any derivation Δ and node n:(a,x) in Δ:

- *Body/output (bo)*: $\neg a \notin out(H_n,a)$, where H_n is the set of nodes of the subtree determined by n;

- *Body/materialisation (bm)*: $\neg a \notin Cn(m(H_n))$, where again H_n is the set of nodes of the subtree determined by *n;*

- *Body/head (bh)*: $\neg a \notin Cn(h(H_n))$ for each set H_n in the dependency family of *n;*

- *Body/fulfilment (bf)*: $\neg a \notin Cn(f(H_n))$ for each H_n in the dependency family of *n*.

We say that a derivation Δ satisfies a constraint iff every node of the derivation does so.

Remark. Care should be taken when considering the *bo* constraint. Its definition is rather more subtle than the others. Whether Δ satisfies the *bo* constraint at node *n*:(*a*,*x*) depends not only on the nodes *actually used in getting n* (or even those used later in the derivation under consideration), but also on the *totality of rules allowed*. We need to check whether *a* is consistent with $out(H_n,a)$, where *out* is an unrestricted output operation (normally the one to be constrained). For the other constraints, we need only check the consistency of *a* with the materialisations, heads, or fulfilments of nodes actually used.

These consistency constraints could also be expressed 'locally', by requiring the body of each node to be consistent with a label attached to that node. The labels may be defined inductively from leaves to root, accumulating information as they grow. Indeed in earlier publications on the *bh* and *bf* constraints, cited above, the authors proceeded in that manner. However, our present view is that the use of labels is best seen as a possible *technique* for checking satisfaction of a constraint, while our present focus is on the concepts that give rise to them.

5. Immediate relations between consistency constraints

Observation 2. *bm* ⇒ *bo* and *bf* ⇒ *bh*.

Clarification. This observation should be understood as saying that in any derivation Δ, and at any node *n*:(*a*,*x*) of Δ, satisfaction of the *bm* (resp. *bf*) constraint at node *n* implies satisfaction of the *bo* (resp. *bh*) constraint at that node, where the constraints themselves are understood as at the end of section 4.

Proof. Immediate from Observation 1 and the understanding that *bo*, *bm* are applied with unsplit dependence, whereas *bh*, *bf* are applied with split dependence.

However, the converse implications, and also both of those between *bm* and *bh*, may fail, as illustrated by the following examples.

Example 6. *Derivation satisfying bh but failing bf.*

$$\begin{array}{cl} (a,x) \qquad (b,x) & \\ \hline (a\vee b,\, x) & \text{OR} \\ | & \text{SI} \\ ((a\vee b)\wedge(a\to\neg x),\, x) & \end{array}$$

This satisfies *bh*: e.g. for the root node, the body $(a\vee b)\wedge(a\to\neg x)$ is consistent with the set of heads in each branch. But no derivation with the same root and a subset of the leaves satisfies fails the fulfilment check: there is no derivation of the root from a single one of the leaves, and body $(a\vee b)\wedge(a\to\neg x)$ is inconsistent with the fulfilment of the leaf (*a*,*x*).

Example 7. Derivation satisfying bm but failing bh.

$$\cfrac{\cfrac{\cfrac{(a,x\wedge y)}{(a,x)}\,\mathrm{WO} \qquad (b,x)}{(a\vee b,x)}\,\mathrm{OR}}{((a\vee b)\wedge(\neg x\vee\neg y),\, x)}\,\mathrm{SI}$$

This satisfies *bm*: e.g. for the root node, body $((a\vee b)\wedge(\neg x\vee\neg y)$ is consistent with materialisations $a\rightarrow x\wedge y$ and $b\rightarrow x$ taken together. But it (and any other derivation from a subset of the same generators to the same conclusion) fails the *bh* check: root body $(a\vee b)\wedge(\neg x\vee\neg y)$ is inconsistent with leaf head $x\wedge y$.

Example 8. Derivation satisfying bo (defined in terms of basic output) and bh, but failing bm. A single example suffices to do both jobs. Put $G = \{(a,\ x\wedge(a\rightarrow y)),\ (\neg a,\ x\wedge(\neg a\rightarrow y))\}$. Consider the derivation:

$$\cfrac{\cfrac{\cfrac{(a,\ x\wedge(a\rightarrow y))}{(a,x)}\,\mathrm{WO} \qquad \cfrac{(\neg a,\ x\wedge(\neg a\rightarrow y))}{(\neg a,x)}\,\mathrm{WO}}{(t,x)}\,\mathrm{OR}}{(\neg y,x)}\,\mathrm{SI}$$

The derivation passes the *bo* constraint when the latter is defined in terms of basic output. In particular, for the root node, $y \notin out_2(G,\neg y)$. To check this, use the semantic definition of out_2 (section 1): putting say $v(a) = 1$ and $v(y) = 0$, we have $\neg y \in V$ but $G(V) = \{x\wedge(a\rightarrow y)\}$ which does not classically imply y.

The derivation also passes *bh* (recalling that this constraint is defined with split dependence). In particular, for the root node we have that for each branch of the tree, $\neg y$ is consistent with the set of all heads on that branch.

But it fails *bm* (recalling that this is defined with unsplit dependence). In fact, any derivation with the same root and a subset of the leaves will fail *bm*. On the one hand, if the derivation uses only one of the leaves, then it cannot yield the root, even without applying consistency constraints; recall from section 1 that $(a,x) \in out_i(G)$ implies $x \in Cn(\{a\}\cup m(G))$, i.e. $a\rightarrow x \in Cn(m(G))$, while clearly in the example $\neg y\rightarrow x$ is not a classical consequence of the materialisation of any one leaf taken alone. On the other hand, if the derivation uses both leaves, then *bm* fails because $\neg y$ is inconsistent with the materialisations of the leaves taken together.

The derivations in examples 6, 7, 8 all make use of the rule OR. In the next section we shall show that this is no accident.

6. Consistency checks in derivations without OR

For derivations that do not use the rule OR, it is immediate that no dependency splits arise (although the tree still branches at every application of AND and of CT). For this reason, one could call these derivations ‘one-track’. For these derivations, it is thus immediate that the two concepts of dependence coincide: a node (a,x) depends on the set of all nodes in the subtree that it generates. Thus for derivations without OR, we immediately have the implications $bf \Rightarrow bh \Rightarrow bm \Rightarrow bo$, by Observation 1. More surprisingly, in this case we also have the converse implications, as we now show.

Observation 3. For derivations not using OR (i.e. using at most the rules SI, AND, WO, CT) the consistency checks *bo*, *bm*, *bh*, *bf* are equivalent at every node.

Clarification. We recall from section 4 that care should be taken when considering the *bo* constraint, as its definition is relative to a choice of unrestricted output operation. In the above observation, it is understood as defined with respect to any one of the four kinds of unrestricted output recalled in section 1, i.e. out_i for $i \in \{1,2,3,4\}$, that allows at least the rules used in whatever derivation is under consideration.

Proof. Knowing already that $bf \Rightarrow bh \Rightarrow bm \Rightarrow bo$, it remains to show $bo \Rightarrow bf$. Fix any derivation tree. For each node $n{:}(a,x)$ in the derivation, write H_n for the set of all nodes in the subtree determined by n. Write $f(H_n)$ for the set of all fulfilments of elements of H_n. It suffices to show that for every node $n{:}(a,x)$ of the tree, if a is inconsistent with $f(H_n)$ then $\neg a \in out(H_n,a)$. As $out(H_n,a)$ is closed under consequence for all four unrestricted output operations, it suffices to show that $\{a\}\cup out(H_n,a) + f(H_n)$. This we now do by induction.

Basis: Suppose $n{:}(a,x)$ is a leaf of the tree. Then $x \in out(H_n,a)$, $f(H_n) = \{a\wedge x\}$, so clearly $\{a\}\cup out(H_n,a) + f(H_n)$.

SI: Suppose $n{:}(a,x)$ is derived by SI from $p{:}(b,x)$. Then $a + b$, $x \in out(H_n,a)$, $out(H_p,b) \subseteq out(H_n,a)$, $f(H_n) = \{a\wedge x\}\cup f(H_p)$. By the induction hypothesis, $\{b\}\cup out(H_p,b) + f(H_p)$. Putting these together, $\{a\}\cup out(H_n,a) + f(H_n)$, as desired.

AND: Suppose $n{:}(a,z)$ is derived by AND from $p{:}(a,x)$ and $q{:}(a,y)$. Then $z = x\wedge y \in out(H_n,a)$, $out(H_p,a) \subseteq out(H_n,a)$, $out(H_q,a) \subseteq out(H_n,a)$, and $f(H_n) = \{a\wedge z\}\cup f(H_p) \cup f(H_q)$. By the induction hypothesis, $\{a\}\cup out(H_p,a) + f(H_p)$ and $\{a\}\cup out(H_q,a) + f(H_q)$. Putting these together (indeed without needing $z = x\wedge y$) we have $\{a\}\cup out(H_n,a) + f(H_n)$.

WO: Suppose $n{:}(a,x)$ is derived by WO from $p{:}(a,y)$. Then $y + x$, $x \in out(H_n,a)$, $out(H_p,a) \subseteq out(H_n,a)$, $f(H_n) = \{a\wedge x\}\cup f(H_p)$. By the induction hypothesis, $\{a\}\cup out(H_p,a) + f(H_p)$. Putting these together (indeed without needing $y + x$) we have $\{a\}\cup out(H_n,a) + f(H_n)$.

CT: Suppose n:(a,x) is derived by CT from p:(a,y) and q:($a \wedge y$,x). Then $x \in out(H_n,a)$, $y \in out(H_p,a) \subseteq out(H_n,a)$, and $f(H_n) = \{a \wedge x\} \cup f(H_p) \cup f(H_q)$. Also $out(H_q,a \wedge y) \subseteq out(H_n,a \wedge y) \subseteq out(H_n,a)$, the last inclusion since as noted we also have $y \in out(H_n,a)$. By the induction hypothesis, $\{a\} \cup out(H_p,a) + f(H_p)$ and $\{a \wedge y\} \cup out(H_q,a \wedge y) + f(H_q)$. Putting these together we have $\{a\} \cup out(H_n,a) + f(H_n)$.

7. Further contexts in which *bo* is equivalent to *bm*

From Observation 3 we know that in the context of simple-minded output, the *bo* and *bm* constraints are equivalent at any node of a derivation. We now show that these two are also equivalent in a number of other contexts.

We begin by going back to Example 8, which showed that in the context of basic output, they are not equivalent: *bo* holds but *bm* fails. If in that example the *bo* constraint is defined in terms of basic reusable output, then *bo* fails in it, for $y \in out_4(G,\neg y)$. This can be checked using the semantic definition of basic reusable output from section 1: if $\neg y \in V$ and e.g. $a \in V$, then $x \wedge (a \rightarrow y) \in G(V) \subseteq V$, so that $(a \rightarrow y) \in V$ and hence $\neg a \in V$; so $G(V) = \{x \wedge (a \rightarrow y), x \wedge (\neg a \rightarrow y)\}$, and thus $y \in Cn(G(V))$. Similarly when $\neg a \in V$.

On an intuitive level, this rather subtle situation may be explained as follows. In the absence of any rules authorising inputs to reappear as outputs or outputs to be reused as inputs, the two are ‘totally isolated’ from each other. Input messages pass into G, which puts out whatever it likes, and without passing it back to the input side. So if a is an input and also happens to reappear as the antecedent of a conditional in the output, then the machine has no authority to perform a detachment, whether in the output or in the output.

This suggests that if we add either CT or the identity rule to those for basic input, then the *bo* constraint may become equivalent to *bm*. That is indeed the case.

Observation 4. When the *bo* constraint is defined in terms of reusable output, it is equivalent to the *bm* constraint, for any a and H.

Proof. We need to show that $\neg a \in Cn(m(H))$ implies $\neg a \in out_4(H,a)$ for any a,H, the converse being given already by Observation 2. Suppose $\neg a \in Cn(m(H))$, i.e. that a is classically inconsistent with $m(H)$. Then by Observation 6 of (Makinson and van der Torre, this volume), $\text{out}_4(H,a) = Cn(H(L)) = Cn(h(H))$, where L is the set of all formulae and $h(H)$ is the set of all heads of elements of H. But by classical logic $Cn(m(H)) \subseteq Cn(h(H))$, so $\neg a \in out_4(H,a)$ and we are done.

We recall (Makinson and van der Torre, this volume) that the identity rule puts (a,a), for every formula a, into the output of every generating set. Thus given an output operation out_i the system defined by adding the identity rule may be defined as $out_i(G \cup I)$ where $I = \{(y,y)$: y a formula$\}$.

Observation 5. When the *bo* constraint is defined in terms of basic output plus the identity rule, it is equivalent to the *bm* constraint, for any a and H.

Proof. From Observation 2 and classical logic, $\neg a \in out_2(H \cup I, a)$ implies $\neg a \in Cn(m(H))$ for any a,H, so we verify the converse. We show more generally that $x \in Cn(m(H) \cup \{a\})$ implies $x \in out_2(H \cup I, a)$. Suppose $x \in Cn(m(H) \cup \{a\})$; we use the semantic characterisation of out_2 (section 1). Let V be any complete set containing a. We need to show that $x \in Cn(H^+(V))$ where $H^+ = H \cup I$. By the supposition, it suffices to show that $m(H) \cup \{a\} \subseteq Cn(H^+(V))$.

We have $a \in Cn(H^+(V))$ since $(a,a) \subseteq I \subseteq H^+$ and $a \in V$. Now let $(b,y) \in H$; we need to check that $b \rightarrow y \in Cn(H^+(V))$. If $b \in V$ then since $(b,y) \in H \subseteq H^+$ we have $y \in H^+(V)$ so $b \rightarrow y \in Cn(H^+(V))$. On the other hand, if $b \notin V$ then since V is complete we have $\neg b \in V$ so since $(\neg b, \neg b) \subseteq I \subseteq H^+$ we have $\neg b \in H^+(V)$ so again $b \rightarrow y \in Cn(H^+(V))$.

Observation 6. When the *bo* constraint is defined in terms of basic output plus both reusability and the identity rule, it is equivalent to the *bm* constraint, for any a and H.

Proof. We know from (Makinson and van der Torre, this volume) that the system constituted by basic output plus both the reusability and identity rules collapses into classical consequence, i.e. $x \in out(G,a)$ iff $x \in Cn(m(G) \cup \{a\})$. Then $\neg a \in out(G,a)$ iff $\neg a \in Cn(m(G) \cup \{a\})$ iff $\neg a \in Cn(m(G))$.

Combining Observations 3 -6, we see that the *bo* constraint is equivalent to *bm* for all definitions of the former in terms of out_i studied in this paper, except $i = 2$ (basic output). This may conveniently be expressed in a table.

Kind of output	**bo = bm ?**	**Proof**
Simple-minded	yes	Observation 3
Simple-minded + CT	yes	Observation 3
Simple-minded + Identity	yes	Observation 3
Simple-minded + CT + Identity	yes	Observation 3
Basic	no	Example 8
Basic + CT	yes	Observation 4
Basic + Identity	yes	Observation 5
Basic + CT + Identity	yes	Observation 6

8. The role of root and leaves in a derivation

When carrying out a consistency check on a derivation Δ, do we need to verify it for every node in Δ, or *does it suffice to consider the root node*? And if we are considering a node m:(b,y), must we check the consistency of b with respect to *all* nodes on which n depends, or is it enough to consider *only those that are leaves*?

These questions are significant computationally. They are also conceptually important if we are interested in providing a semantics for constrained output operations. This is because any semantics will have to abstract from the intermediate structure of derivations, retaining only information about the leaves and the root.

It is important to note that the above questions may be posed at two different levels, of *specific derivations*, and of *derivability*. Let us say that a constraint may be restricted in a certain way *without loss of force for derivations*, iff whenever a derivation Δ satisfies the constraint in its restricted form, it satisfies it fully. On the other hand, we say that a constraint may be restricted in a certain way *without loss of force for derivability*, iff whenever a derivation Δ satisfies the constraint in its restricted form, there is some derivation Δ' of the same root and the same (or fewer) leaves that satisfies the constraint fully. Evidently, the former implies the latter, but the converse may fail.

In this section we give some straightforward results for derivations, and in the following section some more difficult ones that hold only for derivability.

Observation 7. For each of the output operations out_i (i = 1,2,3,4) the *bf* constraint may be applied to the root only without loss of force for derivations.

Proof. Let Δ be any derivation with root n:(a,x). Let m:(b,y) be any node of Δ, and suppose that for some dependency set H_m of m, $f(H_m) + \neg b$. We need to show that for some dependency set H_n of n, $f(H_n) + \neg a$. Since m is a node in the tree Δ of which n is the root, there is a dependency set H_n of n with $m \in H_n$ and $H_m \subseteq H_n$. Hence $b \wedge y \in f(H_n)$ and $f(H_m) \subseteq f(H_n)$. Thus $f(H_n) + b \wedge \neg b + \neg a$ and we are done.

Observation 8. For each of the output operations out_i (i = 1,2,3,4), and at any node of any derivation, the *bo*, *bm*, *bh* constraints may be applied with respect to leaves only without loss of force for derivations.

Proof. Let Δ be any derivation and let m:(b,y) be any node in Δ. Let H_m be any dependency set of m, and let $L(H_m)$ be the set of leaves in H_m.

For the *bo* constraint, we need to show that if $\neg b \notin out(L(H_m),b)$ then $\neg b \notin out(H_m,b)$, i.e., if $(b,\neg b) \notin out(L(H_m))$ then $(b,\neg b) \notin out(H_m)$. Noting that $L(H_m) \subseteq H_m \subseteq out(L(H_m))$ (the latter since dependency is unsplit for the *bo* constraint) and recalling that $out(G)$ is a closure operation and so satisfies cumulativity, $out(L(H_m)) = out(H_m)$ and we are done.

For the *bm* constraint, we need only show that if $m(H_m) + \neg b$ then $m(L(H_m)) + \neg b$. It suffices to show $m(L(H_m)) + m(H_m)$. Recalling that for the bm constraint dependency is unsplit, we note by inspection that the material counterparts of the rules defining output are all classically correct. In other words, for each rule, the materialisations of the premises of the rule together classically imply the materialisation of the conclusion; for example, for CT, the formulae $a \rightarrow x$ and $a \wedge x \rightarrow y$ together classically imply $a \rightarrow y$.

For the *bh* constraint, we need to show that if $h(H_m) + \neg b$ then $h(L(H_m)) + \neg b$. It suffices to show $h(L(H_m)) + h(H_m)$. Essentially the same argument can be used as for the *bm* constraint taking account, however, of split dependency for the *bh* constraint. We check by inspection that for each rule defining output, the heads of the premises of the rule together classically imply the head of the conclusion, and moreover, in the special case of the rule OR, the premise heads taken separately imply it (indeed, they are identical with it).

We note in passing, as a corollary, that we may without loss of force redefine the notion of dependency so that when a node is not a leaf then it is excluded from its dependency sets. For the *bo*, *bm*, *bh* constraints this is immediate from Observation 8. For the *bf* constraint, it suffices to note that that for each rule defining output, the fulfilments of the premises of the rule imply their heads, which as we have just observed in the last part of the proof of Observation 8, together classically imply the head of the conclusion, and moreover do so separately in the special case of the rule OR.

Observation 9. For simple-minded output (and its extensions by CT and/or I) the application of any one of the consistency constraints *bo*, *bm*, *bh*, *bf* may be restricted to the root only, with respect to the leaves only, without loss of force for derivations.

Proof. We use observations 7, 3, and 8. By Observation 7, *bf* may be applied wlf (without loss of force for derivations) to the root node only; so since OR does not occur in the derivation, we know by Observation 3 that the same is true for *bo, bm, bh*. Hence by Observation 8, for each of *bo*, *bm*, *bh*, application may be restricted wlf to the root node only, with respect to leaves only. Hence by Observation 3 again (strictly speaking, re-running its proof using L_n rather than H_n) the same is also true of *bf*, and we are done.

Observation 9 permits a semantic characterisation of simple-minded output constrained by any of *bo*, *bm*, *bh*, *bf*, and likewise for its extensions by CT and I. We recall that out_1 is simple-minded output, out_3 is its extension by CT, and we write out_{1+}, out_{2+} to indicate their extensions by the identity rule I. When out_i is an unconstrained operation, we write out_i^c for its restricted version under any of the constraints *bo, bm*, *bh*, *bf*.

Observation 10. Semantic characterisations of constrained output operations without OR. For $i = 1,3,1^+,3^+$ we have: $(a,x) \in out_i^c(G)$ iff for some $H \subseteq G$, $(a,x) \in out_i(H)$ and $f(H) +/ \neg a$,

Proof. Recall that dependency is unsplit in derivations without OR. For the left to right implication simply put H to be the set of leaves of a constrained derivation of (a,x) from G, and apply Observation 3. For the right to left implication, apply Observation 9, noting that each operation $out_i^c(G)$ is trivially monotonic in G.

9. Constraints and derivability

Unfortunately, Observation 7 fails for the constraints *bo*,*bm*,*bh*, which may lose force for derivations using OR, when applied to the root only.

Example 9. Derivation satisfying bo,bm,bh applied to the root, but not when applied fully.

$$\dfrac{\dfrac{(a,x) \qquad \dfrac{(x,\neg(a\wedge x))}{(a\wedge x,\neg(a\wedge x))}\ \text{SI}}{(a,\neg(a\wedge x))}\ \text{CT} \qquad (x,\neg(a\wedge x))}{(a\vee x,\neg(a\wedge x))}\ \text{OR}$$

The body $a\vee x$ of the root is consistent with the heads of all the nodes taken together, so that all three constraints hold at the root. However, the body a of the intermediate node $(a,\neg(a\wedge x))$ is not consistent with the materialisations of the two leaves on which it depends, so that the *bm*, *bh* constraints fail when applied fully. The *bo* constraint also fails at the same node, if defined in terms of reusable output, since $\neg a \in out_4(H,a)$ where H is the set of nodes on which it depends.

Moreover, it can be shown that the failures carry over to derivability. In other words:

Observation 11. There is no derivation with the same root as that of Example 9, and with the same (or fewer) leaves, that satisfies any of the *bo,bm,bh* constraints applied fully.

Outline of proof. We make an exhaustive search for such a derivation. First, we observe that the root $(a\vee x,\neg(a\wedge x))$ cannot be derived, even without constraints, using only one of the leaves (a,x) or $(x,\neg(a\wedge x))$, since the materialisation of the root is not a classical consequence of the materialisation of the leaf (section 1). Second, if the two premises are combined in a derivation then the first proof rule combining them is either CT (two ways), AND or OR. The application of CT shown in Example 9 and the application of AND are blocked by consistency checks, the second application of CT and OR only derive trivial consequences, respectively $(a\wedge x,x)$ from $(a\wedge x,t)$ and $(a\wedge x,x)$, and $(a\vee x,t)$ from (a,t) and (x,t). Consequently there is no derivation of the root $(a\vee x,\neg(a\wedge x))$ from the leaves (a,x) and $(x,\neg(a\wedge x))$ satisfying any of the *bo,bm,bh* constraints.

Again, Observation 8 fails for *bf*, which may lose force for derivations when it is applied with respect to leaves only.

Example 10. Derivation satisfying bf wrt leaves only, but not when applied fully.

```
(t,x)           (t,x)
  |      SI       |
(a,x)             |
------------------   OR
      (t,x)
        |   SI
     (¬a,x)
```

Here, the body of each node is consistent with the fulfilment of the leaves, but the body $\neg a$ of the root node is not consistent with the fulfilment of the intermediate node (a,x) on which it depends.

However, it is clear that this derivation is a ‘funny’ one. Although it fails the *bf* constraint, there is clearly *another* derivation of the same root node from the same leaf that satisfies the constraint. It can be shown that this is always the case.

Observation 12. For every derivation Δ there is a derivation Δ' with the same root and the same leaves, such that if Δ satisfies the *bf* constraint wrt leaves only, then Δ' satisfies the *bf* constraint fully.

Clarifications and remarks. This observation is understood modulo each of the output operations out_i (i = 1,2,3,4) and their extensions with the rule I. ‘The same’ means identity up to classical equivalence, in accord with the understanding (section 1) that replacement of equivalent propositions is treated as a ‘silent rule’. We give two proofs, with different values of Δ'. The first proof transforms the derivation tree; the second merely changes its labels, indeed only the bodies of the labels.

Outline of first proof. We sketch an algorithm that explicitly constructs Δ' from Δ. We would like to rewrite derivations so that SI is applied only in the final step of Δ', because in that case the fulfilments of the leaves imply the fulfilments of all other nodes (except for that of the root, derived by SI). This can be done for the derivation in Example 9.

```
(a,x)          (b,x)         |  (a,x)          (b,x)
  |   SI         |           |  ------------------   OR
(a∧c,x)          |           |       (a∨b,x)
----------------   OR        |          |      SI
 ((a∧c)∨b,x)                 |     ((a∧c)∨b,x)
```

However, sometimes we cannot rewrite derivations so that SI is applied only at the last step because, as remarked in Observation 18 of (Makinson and van der Torre, this volume) no rewrite rules are known for reversing applications of SI/CT and SI/AND. Fortunately, there are weaker rewrite rules that suffice for Observation 12. The crucial rewrite rules are the following one for SI/CT and an analogous one for SI/AND.

$\begin{array}{ccc} (a,x) & (c,y) & \\ \mid \ \text{SI} & \mid \ \text{SI} & \\ (a\wedge b,x) & (c\wedge d,y) & \\ \hline \multicolumn{2}{c}{(a\wedge b,y)} & \text{CT: } a\wedge b\wedge x \approx c\wedge d \end{array}$	$\begin{array}{ccc} (a,x) & (c,y) & \\ \mid \ \text{WO} & \mid & \\ (a,x\vee c) & \mid \ \text{SI} & \\ \mid \ \text{SI} & \mid & \\ (a\wedge(\neg x\vee c),x\vee c) & (a\wedge c,y) & \\ \hline \multicolumn{2}{c}{(a\wedge(\neg x\vee c),y)} & \text{CT} \\ \multicolumn{2}{c}{\mid \ \text{SI}} & \\ \multicolumn{2}{c}{(a\wedge b,y)} & \end{array}$

In the left derivation the fulfilments of the leaves together with the body of the root do not necessarily imply the fulfilments of all nodes, but in the right derivation they do so. SI is still applied before CT, but in a ‘minimal’ way: the proposition $\neg x\vee c$ introduced by the rule is implied by the fulfilment of a leaf, namely (c,y), on which depends the node obtained by the following application of CT, namely $(a\wedge(\neg x\vee c),y)$.

In general, a set of rewrite rules (the two above together with SI/SI ⇒ SI, SI/WC ⇒ WC/SI, and SI/AND ⇒ SI/AND/SI) creates a derivation Δ′, for which an inductive argument shows that the fulfilment of every node other than the root is implied by the fulfilments of the leaves of each dependency set of its child. Hence the fulfilment of every node is implied by the body of the root together with the fulfilments of the leaves of each of its dependency sets.

Second proof. We change the labels on Δ, and then extend it by just one more node. The basic idea is to put a ceiling on the bodies of nodes, to make sure that application of SI never makes them stronger than is necessary to derive the root of the tree from the leaves. No change is made to heads of nodes or to the structure of the tree.

Let m:(b,y) be any node of Δ. Replace b by $b\vee\varphi(m)$ where $\varphi(m) = \vee\{\wedge f(L(H)): H \in D_m\}$. Here D_m is the family of all dependency sets H of Δ that contain m, $L(H)$ is the set of all leaves in H, $f(L(H))$ is the set of fulfilments of elements of $L(H)$, $\wedge$ is conjunction, and $\vee$ is disjunction.

For the additional node, let n:(a,x) be the root of Δ. By the above, it is relabeled as n: $(a\vee\varphi(n),x)$. Add as its child a node n+1, serving as the new root node, and give it the label (a,x), i.e. the same label as for the root node of Δ.

We claim that Δ′ is a derivation of (a,x) from the same leaves (up to logical equivalence) as in Δ, and that it satisfies the *bf* consistency constraint applied fully.

For each leaf $m{:}(b\vee\varphi(m),y)$ of Δ' we have $b \approx b\vee\varphi(m)$ since when m is a leaf and $H \in D_m$, then $(b,y) \in L(H)$ so that $\varphi(m)$ classically implies b.

The last step of Δ' is justified by SI. An induction shows that all other steps in Δ' are justified by the same rules as their counterparts in Δ, so that Δ' is a genuine derivation. The cases for SI, WO, AND, are straightforward. For OR, the essential point is to observe that when node m in Δ is obtained from nodes p and q by OR, then $D_m = D_p \cup D_q$ so that $\varphi(m) \approx \varphi(p)\vee\varphi(q)$. The only delicate case is that for CT, which we give in full. Suppose that in Δ, node $m{:}(b,z)$ is obtained by CT from $p{:}(b,y)$ and $q{:}(b\wedge y,z)$. We need to show that $m{:}(b\vee\varphi(m),z)$ is obtainable by CT from $p{:}(b\vee\varphi(p),y)$ and $q{:}((b\wedge y)\vee\varphi(q),z)$. It suffices to show $(b\wedge y)\vee\varphi(q) \approx (b\vee\varphi(p))\wedge y$. The nodes p and q are elements of the same dependency sets of Δ, so $\varphi(p) = \varphi(q)$. Hence it suffices to show that $\varphi(p)$ implies y. Given the definition of $\varphi(p)$, it suffices to show that for every dependency set H of Δ, $f(L(H))$ implies the head of every node in H. But this holds in any derivation: $f(L(H))$ immediately implies $h(L(H))$ which, as we have already seen in the proof of Observation 8 (case of the *bh* check), implies $h(H)$ and we are done.

It remains to check that Δ' satisfies the full *bf* constraint. By Observation 7 it suffices to show that it does so at the root $n{+}1{:}(a,x)$, i.e. that a is consistent with $f(H')$ for every dependency set H' of Δ'. Let H' be a dependency set of Δ'. Then $H = H' - \{n{+}1\}$ is a dependency set of Δ with $L(H) = L(H')$. Since Δ satisfies the *bf* constraint with respect to leaves, we know that a is consistent with $f(L(H))$. As already noted, $f(L(H'))$ implies the head of every node in H'. To complete the proof, it suffices to show that $f(L(H))$ implies the body of every node in H' other than the root node $n{+}1{:}(a,x)$, i.e. that $f(L(H))$ implies the relabeled body $b\vee\varphi(m)$ of every node m of Δ. But this is immediate from the definition of $\varphi(m)$ and we are done.

10. Open Questions

From Example 9 and Observation 11, we see that in general (for output operations allowing OR), the *bo*, *bm*, *bh* constraints may lose force for derivability if applied to the root only, which dashes hopes for a semantic characterisation of basic or reusable output along the lines of Observation 10. It leaves open the question whether such semantics could be obtained in a different matter, but the outlook is not encouraging.

In the case of the *bf* constraint, prospects seem rather brighter. Putting Observation 7 together with Observation 12 (strictly speaking, with the proof of the latter) we obtain the following.

Observation 13. For each of the output operations out_i (i = 1,2,3,4) the *bf* constraint may be applied to the root only, with respect to leaves only, without loss of force for derivability.

Outline of proof. Let Δ be a derivation that satisfies the *bf* check applied to the root only with respect to leaves only. By the same argument as used for Observation 12, there is a derivation Δ' with the same root and the same leaves, that satisfies the *bf* constraint applied to the root only but with respect to its entire dependency sets. By Observation 7, Δ' satisfies the *bf* check applied fully.

However, there is still some distance between Observation 13 and a semantic characterisation of basic or reusable output with the *bf* constraint. The reason is that in the presence of OR, the *bf* constraint is applied with split dependence; and the pattern of splitting depends not only on the root and the leaves, but also on the places where OR happens to be applied in the derivation tree. So we need to check the consistency of the body of the root, not with the set of fulfilments of all the leaves, but rather with each of certain subsets of that set, and the identity of those subsets depends on the internal structure of the derivation. The question of a semantic characterisation of basic and reusable output under the *bf* constraint thus remains open, with a prospect perhaps brighter than for the other constraints.

In general, our investigations into constrained output have set out from the syntactic end, considering consistency checks on nodes in derivations and looking for semantic counterparts. It would also be interesting to proceed in the reverse direction: considering semantic restrictions on the definitions of output, and seeing how far they can be represented syntactically.

References

Belzer, Marvin, 1987. Legal reasoning in 3-D. In *Proceedings of the First International Conference in Artificial Intelligence and Law*, 155-163. ACM Press: Boston.

Makinson, David, 1999. On a fundamental problem of deontic logic. In *Norms, Logics and Information Systems. New Studies in Deontic Logic and Computer Science*, edited by Paul McNamara and Henry Prakken (Amsterdam: IOS Press, Series: Frontiers in Artificial Intelligence and Applications, Vol 49, 1999) pp 29-53.

Makinson, David, and Leendert W.N. van der Torre. Input/output logics. In this volume.

van der Torre, Leendert W.N., 1997. *Reasoning about Obligations: Defeasibility in Preference-Based Deontic Logic*. Ph.D. thesis, Erasmus University of Rotterdam. Tinbergen Institute Research Series n° 140. Thesis Publishers: Amsterdam.

van der Torre, Leendert W.N., 1998. Phased labeled logics of conditional goals. *Logics in Artificial Intelligence*. Proceedings of the Sixth European Workshop on Logics in AI (JELIA'98). Berlin: Springer, LNCS 1489, pp 92-106.

van der Torre, Leendert W.N., 1999. The logic of reusable propositional output with the fulfilment constraint. In David Basin et al eds, *Labelled Deduction* (Dordrecht: Kluwer).

Acknowledgements

Thanks to Salem Benferhat and anonymous referees for DEON 2000 for comments on drafts. Research for this paper was begun when the second author was working at IRIT, Université Paul Sabatier, Toulouse, France, and at the Max Planck Institute for Computer Science, Saarbrücken, Germany.

David Makinson
Les Etangs B2, Domaine de la Ronce
92410 Ville d'Avray, France
Email: d.makinson@unesco.org

Leendert van der Torre
Department of Artificial Intelligence
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands
Email:torre@cs.vu.nl