DEON'02 SUBMISSION

Authors' Names:
Alan S. Abrahams and Jean M. Bacon

Authors' Affiliation:
University of Cambridge Computer Laboratory

Authors' Address:
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
UK

Authors' Email Addresses:
Primary contact: alan.abrahams@cl.cam.ac.uk
Secondary contact: jean.bacon@cl.cam.ac.uk

Article Title:
The Life and Times of Identified, Situated, and Conflicting Norms

Abstract:
In this paper we argue for a treatment of obligations, permissions, and prohibitions that differs from the standard treatment of these notions in deontic logic. Firstly, we propose that instantiated norms be treated as individual, *identified* entities - that is, variables that can be quantified over - rather than simply as logical operators as in Standard Deontic Logic. This allows us to refer to specific instances of obligations, permissions, and prohibitions. We explain why we believe that norms take, as their arguments, sets of occurrences, rather than simply propositions as in the standard treatment. Further, we argue that specific, identified norms themselves are brought about by occurrences. We provide an account of the *life-cycle* of norms: we explain how individual identified norm-instances are generated from general norms through functions of occurrences, and how each such instance's life may end with its fulfillment, violation, or nullification. In addition, we suggest that norms are *situated*: that they must be tagged with the context in which they were written or spoken. This is necessary for *conflict specification, detection, and resolution* purposes. Finally, we tag our conclusions with a *time*, so that, without contradiction, we may non-monotonically conclude different results as cases and norms vary over time.

The Life and Times of Identified, Situated, and Conflicting Norms

1 Introduction

In this paper we argue for a treatment of obligations, permissions, and prohibitions that differs from the standard treatment of these notions in deontic logic. Firstly, in Section 2, we propose that instantiated norms be treated as individual, *identified* entities - that is, variables that can be quantified over - rather than simply as logical operators as in Standard Deontic Logic. This allows us to refer to specific instances of obligations, permissions, and prohibitions. We explain why we believe that norms take, as their arguments, sets of occurrences, rather than simply propositions as in the standard treatment. Further, we argue that specific, identified norms themselves are brought about by occurrences. In Section 3, we provide an account of the *life-cycle* of norms: we explain how individual identified norm-instances are generated from general norms through functions of occurrences, and how each such instance's life may end with its fulfillment, violation, or nullification. In addition, we suggest (Section 4) that norms are *situated*: that they must be tagged with the context in which they were written or spoken. This is necessary for *conflict specification, detection, and resolution* purposes (Section 5). Finally, in Section 6, we tag our conclusions with a *time*, so that, without contradiction, we may non-monotonically conclude different results as cases and norms vary over time.

2 Identity

In this section we describe what we mean by an identified occurrence, as opposed to a proposition or a logical operator, and how this notion is used for representing and assessing obligations. We then describe how specific identified obligations may be brought about from general obligations through functions on occurrences. In the spirit of Jones and Sergot (1993), we will use a library lending scenario throughout, though we will make use of our own specific set of regulations and circumstances.

2.1 Representing and Assessing Obligations with Occurrences

Standard Deontic Logic presumes that being obliged to do that which is prohibited is a logical contradiction. Obligation and prohibition are treated as operators, typically with the interdefinability axiom:

$$P\alpha \equiv_{def} \neg O \neg \alpha$$

meaning that if a state-of-affairs, α, is permitted then it is not obliged that not α. In contrast, we take as our starting point the assumption that permissions and obligations are independent entities - variables that are quantified over - and that conflicting norms can exist. Our view is that being obliged to do that which is forbidden is not a logical contradiction, but a choice dilemma. The choice may involve deciding which identified directive to violate, or deciding which to regard as void in the circumstance (that is, in each case of application by a norm interpreter).

Before we investigate the representation of an identified norm further, we need to introduce the notion of an occurrence, as distinct from a proposition. A proposition may be associated with a truth value: letting `A` be the proposition 'John returned the book', we assess that `A` is true in the case that there are one or more occurrences of John having returned the book. In contrast, an occurrence is an identified entity which occupies a moment or interval of time and takes as its arguments participants in various roles. We might therefore have `returning_1` which takes as its arguments `book_copy_1235` in the role `returned` and `John` in the role `returner`. Assuming John borrowed `book_copy_1235` on two separate occasions following are two separate occurrences of John returning that book:

```
returning_1 (on 3 June 2001)
        returned:       book_copy_1235
        returner:       John (user_id = user896)
returning_2 (on 5 July 2001)
        returned:       book_copy_1235
        returner:       John (user_id = user896)
```

We use Skolem constants to identify specific occurrences of a given type: `returning_1` as an instance of the type `returning`. Each occurrence of returning may be associated with a time: the first return may have

been on 3rd June 2001, whilst the second may have been on 5th July 2001. The proposition A ('John returned book_copy_1235') is strictly true from 3rd June 2001 onwards, since from that moment on it is true that John has at some stage returned the book. A proposition, then, becomes true as a result of one or more identified occurrences. Of identified occurrences we can say only that they happened (or did not): we speak of their existence, rather than of their 'truth'. Since occurrences occupy (perhaps unspecified) moments or intervals of time, they are inherently temporal in nature, whereas propositions - in the absence of extension to cater for time - are atemporal.

The notion of individuating events has its philosophical original in Davidson (1980), and has been examined and substantiated in various forms by, inter alia, Kowalski and Sergot (1986), Bennett (1988), Parsons (1990), Kimbrough (1998), and Pianesi and Varzi (2000). Philosophical subtleties abound in the various uses of the term 'event' in the literature. For instance, Bennett (1988) controversially argues that John's crossing the Channel and John's swimming the Channel are the same 'event', whereas we treat them as separate occurrences. Kimbrough (personal communication) argues that an obligation state, $\exists e$ ought(e), can be the same as a violation state, violating(e); our contention is that each is an independent entity: an occurrence being_obliged_1, and an occurrence violating_1, where the participant in the role violated in violating_1 is the obligation identified as being_obliged_1. To avoid any confusion with the various usages of the word 'event' in the literature, we have chosen the term 'occurrence' to refer to our entities.

We believe the notion of an occurrence, absent from standard treatments of deontic logic, is useful for the representation and assessment of obligations. Brown (2000) speaks of the distinction between simply dischargeable obligations and standing obligations. The latter have been the traditional purvey of deontic logic. As we will illustrate here, it seems that Standard Deontic Logic (SDL) copes poorly with simply dischargeable obligations. This is problematic since such obligations form a large portion of the obligations we wish to reason about in commercial contractual scenarios. Consider the case where John has borrowed book_copy_1235 for the second time on 1st July 2001 and has an obligation - a specific obligation - to return it. Taking A as the proposition that John returns the book, in Standard Deontic Logic we might then say:

OA

meaning 'John is obliged to return book_copy_1235'. How though, do we distinguish this second obligation, from John's previous obligation, which arose when he first borrowed the book? In SDL, the first obligation is also represented as:

OA

Collapsing the obligations together through logical derivation results in dangerous information loss, since originally we had two obligations, but standard propositional logic leaves us with one:

$OA \wedge OA$

——————— (standard propositional logic)

OA[1]

SDL fails to distinguish between propositions about norms, and the actual norms themselves. Makinson (1999) comments that simply describing norms as true or false is insufficient, and that it is a fundamental problem of deontic logic that norms are simply considered to have truth values. We argue that, since propositions about norms are derived from the norms themselves, invalid or misleading inferences will result if we deal merely with the propositions rather than with the identified norms that make those propositions true or false. Assume, on both occasions, John fulfils his obligation to return the book (that is, A is true), we have:

$OA \wedge A \wedge OA \wedge A$

——————————————— (standard propositional logic)

$OA \wedge A$

[1] Similar information loss occurs when we derive such conclusions as OA from $O(A \wedge B)$ in SDL. As SDL does not identify the particular obligation $O(A \wedge B)$ we have no way of determining that OA is in fact a partial description of $O(A \wedge B)$. Maintaining an identifier for the obligation (e.g. using subscripts, $O_1(A \wedge B)$ and $O_1(A)$) would be useful in a database environment to allow us to look up the full description of the obligation O_1 when we have partial information on it.

The two different fulfillment occurrences are not distinguished here. Most problematically, it seems that, had John returned the book only the first time and his dog had eaten it on the second occasion we would still have:

$OA \wedge A$

This says that A was obliged and A was done. Using Anderson's (1958) reduction of obligation to violation in the case of falsity of the obliged proposition ($OA =_{def} \Box(\neg A \rightarrow Violation)$), John's violation of his second obligation is not evident since A is true by virtue of fulfillment of the first obligation.

To rectify the above problems, it seems that norms (and occurrences which fulfil or violate those norms) should be individually identified. A treatment of obligations as entities, rather than as operators on propositions, has been proposed in Kimbrough (2001). Kimbrough has recommended identifying both obligation states (instead of the O operator) and violation states (instead of the insufficiently specific `Violation` predicate) to correct some existing deficiencies of SDL. We wish to justify Kimbrough's suggestion and, using our own notation, illustrate their pertinence to the book-borrowing example.

In our world of identified norms, the particular obligation of, say, John, to return the book he borrowed on 1st June may be written, informally, as:

```
being_obliged_1 (arising on 1st June 2001)
        obliged:        first occurrence, on or after 1st June 2001,
                        of John returning book_copy_1235
```

Ignoring return deadlines for the time being, we have an identified obligation, `being_obliged_1`, where what is obliged is the first occurrence of John returning the book he borrowed. The second obligation may likewise be represented as:

```
being_obliged_2 (arising on 1st July 2001)
        obliged:        first occurrence, on or after 1st July 2001,
                        of John returning book_copy_1235
```

Now, `returning_1` fulfils the first obligation (`being_obliged_1`), whilst `returning_2` fulfils the second (`being_obliged_2`). This can be assessed computationally. Each occurrence can be stored in a database. For instance, one possible schema, a tabular format which has been implemented in Abrahams and Bacon (2001), might be as follows:

Participant	Occurrence	Role
book_copy_1235	returning_1	returned
user896 (John)	returning_1	returner
Query1	being_obliged_1	obliged
book_copy_1235	returning_2	returned
user896 (John)	returning_2	returner
Query2	being_obliged_2	obliged

where:

```
Query1 = an identifier for the root node of the syntax tree for the query:
        SELECT first occurrence, on or after 1st June 2001, of
        John returning book_copy_1235
Query2 = an identifier for the root node of the syntax tree for the query:
        SELECT first occurrence, on or after 1st July 2001, of
        John returning book_copy_1235
```

Using a so-called 'continuous query' mechanism for assessing the changing results of a stored query as new data is added to the database, we can computationally determine that `returning_1` is covered by `Query1` and that `returning_2` is covered by `Query2`. Furthermore, as both queries only ever return a maximum of one occurrence (hence the criterion '`first`' in each query) we see that `returning_1` ***fills*** `Query1` and `returning_2` ***fills*** `Query2`. We can then conclude that `returning_1` ***fulfills*** `being_obliged_1` and `returning_2` ***fulfills*** `being_obliged_2`.

In the case of concurrent obligations, Kimbrough (2001) and Daskalopulu and Sergot (2001) recommend the use of a `sake_of()` predicate to allocate fulfillment occurrences to the obligations they are intended to fulfil. This is necessary since, for instance, a consumer may purchase two similar items in quick succession, and the supplier may have two separate obligations to deliver the item. Considering the case of Peter separately purchasing two pizzas on the same night from Susan, the obligations *cannot* be represented simply as:

```
being_obliged_3
        obliged:        first occurrence, tonight, of Sue delivering pizza
being_obliged_4
        obliged:        first occurrence, tonight, of Sue delivering pizza
```

This is because a delivery of a single pizza, say, `delivery_1`, should be allocated to at most one order, and in the absence of further specification it seems that `delivery_1` of a single pizza could fulfil both `being_obliged_3` and `being_obliged_4` which is not what is intended. Since different allocations are possible over time, and depending on which allocation basis is used (e.g. most-recent-first, least-recent-first, or arbitrarily complex allocation criteria) we might choose to use an occurrence of `allocating` in place of Kimbrough's `sake_of()` predicate. We might then represent the concurrent obligations as:

```
being_obliged_3
        obliged:        first occurrence, tonight, of Sue delivering pizza,
                        and where said occurrence is allocated to being_obliged_3
being_obliged_4
        obliged:        first occurrence, tonight, of Sue delivering pizza
                        and where said occurrence is allocated to being_obliged_4
```

We then require a rule that each delivery of a single pizza must be allocated, using some specified basis, to a single obligation. Following application of such a rule, we might have the following occurrences of allocating:

```
allocating_1
        allocated:              delivery_1
        allocated_to:           being_obliged_3
        basis_of_allocation:    least_recent_first
allocating_2
        allocated:              delivery_2
        allocated_to:           being_obliged_4
        basis_of_allocation:    least_recent_first
```

We can then ensure that each delivery satisfies only a single obligation.

We do not here deal with assignment of performances to multiple obligations, such as when a single delivery of two pizzas (or similarly, a single payment of many dollars) fulfills multiple obligations. In the latter case, each delivered *pizza* (or similarly, paid dollar) is allocated to an obligation, rather than each *delivery* of pizza (similarly, payment of dollars). Neither do we deal with the complexities of accumulation of debts, such as when multiple purchases-on-account during a month are aggregated at month end into a single obligation to pay during the following month, and such obligations may accumulate from month to month. Assignment of payments to purchases, and corresponding conclusions about transfer of ownership for each item purchased during the period is generally controlled by sophisticated organization-specific policies which are outside this paper's scope. In this paper, then, we make the simplifying assumption that each discrete performance occurrence pertains to a single obligation.

Given our above-specified obligations, and our ability to deduce their fulfillment by determining whether queries are filled, we can derive the following occurrences when John returns `book_copy_1235` on time, on both occasions:

```
fulfillment_1
        fulfilled:      being_obliged_1
        fulfiller:      returning_1
fulfillment_2
        fulfilled:      being_obliged_2
        fulfiller:      returning_2
```

It is already evident that the identification of obligations and occurrences brings some benefits over the operator-based account of Standard Deontic Logic: separate obligations can be uniquely identified, and we can determine which of these several obligations has been fulfilled, and by which occurrences of returning, even when the content of the obligation (e.g. 'returning `book_copy_1235`') is similar.

In this section, we have posited the existence of two separate obligations (being_obliged_1 and being_obliged_2) upon John to return the book each time, without explanation of their origin. In the next section, we look at how individual obligation instances may be born from general obligations.

2.2 From General Obligations to Specific Obligations Instances: Functions of Occurrences

Specific obligations instances may be brought about from general obligations through functions on occurrences. In the case of the book borrowing example, it is likely that we have a general rule of the form:

```
Borrowers are obliged to return books borrowed within 14 days.
```

We need some mechanism for generating specific obligations from general specifications of this nature. We choose the device of an identified function, function1, which takes as its domain a set of occurrences, X, and produces as its range a resulting set of obligations, being_obliged_function_1(X). We append the occurrence type produced by the function to the start of the function name to make the output range more clear and more easily accessible to the query mechanism. The domain of the function, being_obliged_function_1, is the set of occurrences of borrowing from our library: that is, roughly, any occurrence in the database having an identifier beginning with borrowing, and having as its lender our library. Assume John borrows the same book on two occasions, giving us:

```
borrowing_1 (on 1 June 2001)
        borrowed:       book_copy_1235
        borrower:       John (user_id = user896)
        lender:         Free Library of America
borrowing_2 (on 1 July 2001)
        borrowed:       book_copy_1235
        borrower:       John (user_id = user896)
        lender:         Free Library of America
```

It should be evident that both these borrowings, borrowing_1 and borrowing_2, are in the domain of being_obliged_function_1. The function then generates two separate obligations, identifiable as being_obliged_function_1(borrowing_1) and being_obliged_function_1(borrowing_2) respectively. These identifiers could be thought of as alternative identifiers, usable instead of the identifiers, being_obliged_1 and being_obliged_2 which we used earlier (§2.1) to identify John's separate obligations. Earlier we assumed that being_obliged_1 and being_obliged_2 were freestanding obligations without provenance in any particular norm or regulation. We can now see, though, how specific obligations are generated, via functions, from stipulated norms and actual occurrences, and we can use such functions to identify each specific obligation instance. The function defining our norm is then:

```
being_obliged_function_1
        domain:         X = any occurrence of borrowing from library
        range:
                being_obliged_function_1(X) where
                        obliged         first occurrence, on or after date of X,
                                        and within 14 days of X,
                                        of borrower in X returning borrowed in X
                                        (to lender in X)[2]
```

[2] In the tabular format shown earlier, this function can be represented and stored as:

```
        Participant             Occurrence                      Role
        -----------             ----------                      ----
        Query_3                 being_obliged_function_1        domain
        Query_|X|_4             being_obliged_function_1(X)     obliged
```

where:

```
        Query_3 = an identifier for the root node of the syntax tree for the query:
                        SELECT occurrences of borrowing from library
                  (Notice that domain of being_obliged_function_1 = X = Query3)
        Query_|X|_4 = an identifier for the root node of the syntax tree for the
                      parameterized query which takes |X| as parameter:
                        SELECT first returning occurrence, on or after |X|.date AND
                               within 14 days of |X|.date WHERE
                               returned = |X|.borrowed AND returner = |X|.borrower
```

(|X| is a bound variable denoting the occurrence that was covered by Query3 and which therefore generated the occurrence identified as being_obliged_function_1(X). For |X| merely substitute the occurrence identifier in the domain of the function. Therefore, for borrowing_1, Query_**|x|**_4 becomes Query_**borrowing_1**_4, and **|x|**.date becomes **borrowing_1**.date.)

Given that our continuous query mechanism can determine that the above-mentioned occurrences of borrowing (`borrowing_1` and `borrowing_2`) fall within the query…

```
SELECT any occurrence of borrowing from Free Library of America
```

… which defines the domain of the function, our two obligations are then the following two identified occurrences:

```
being_obliged_function_1(borrowing_1)
        obliged:        first occurrence, between 1st and 15th June 2001,
                        of John returning book_copy_1235
                        (This query is identified as Query_borrowing_1_4: see footnote 2)
being_obliged_function_1(borrowing_2)
        obliged:        first occurrence, between 1st and 15th July 2001,
                        of John returning book_copy_1235
                        (This query is identified as Query_borrowing_2_4: see footnote 2)
```

`being_obliged_function_1(borrowing_1)` is an obligation instance that arises on 1st June 2001, whilst `being_obliged_function_1(borrowing_1)` is an obligation instance that arises on 1st July 2001.

Here we have shown how identified occurrences and identified norms allow us to separately identify the obligations generated by a norm over time. Derivation of specific identified obligations from general obligations is not treated in SDL.

3 Life Cycle of Norms

In SDL it is not clear that John's first obligation to return the book he first borrowed was fulfilled: $OA \wedge A$ should, intuitively, imply $\neg OA$ since a fulfilled dischargeable obligation no longer stands. Or more specifically, we should be able to infer that the obligation (say, `being_obliged_function_1(borrowing_1)`) once was in force, but is now fulfilled and no further call to action results. We therefore must be aware of occurrences of either its fulfillment, violation, or being voided. For example:

```
fulfillment_1
        fulfilled:      being_obliged_function_1(borrowing_1)
```

or

```
violating_1
        violated:       being_obliged_function_1(borrowing_1)
```

or

```
being_void_1
        voided:         being_obliged_function_1(borrowing_1)
```

Capturing such occurrences allows us to capture the *life-cycle* of an obligation, and thereby assess whether it is still active and requires fulfillment. Standard Deontic Logic allows us to make no such inferences. This is because SDL deals with truth-valued propositions about general standing obligations, rather than with specific identified obligations.

As we have seen, there is a variety of ways in which an obligation may terminate: each of the above occurrences (fulfilling, violating, voiding) counts as a cessation of the obligation. We can define a function to capture this:

```
ceasing_function_1
        domain:         X = any occurrence of fulfilling, violating, or voiding
                            an obligation
        range:
                ceasing_function_1(X) where
                        ceased:         |X|.theme[3]
```

A fulfillment, `fulfillment_100`, of say `being_obliged_1`, would then produce the following cessation occurrence:

```
ceasing_function_1(fulfillment_100)
        ceased:         being_obliged_1
```

[3] `theme` here refers to any of the open set of role names: `fulfilled`, `violated`, `voided` since these domain-specific roles can be generalized to the semantic role 'theme' commonly used in knowledge representation in artificial intelligence (Allen 1995; Sowa 2000).

To capture fulfillment and violation of our obligations in our book-borrowing example we make use of the function device once again. These functions demonstrate how occurrences of fulfilling and violating are produced in each case.

The function that describes the fulfillment conditions for John's 1st obligation (`being_obliged_function_1(borrowing_1)`) is as follows:

```
fulfilling_function_1
        domain:        X = an occurrence of filling where a query describing
                           obliged occurrences is filled (completely).
        range:         fulfilling_function_1(X) where
                               fulfilled:     obligation associated with
                                              the query |X|.filled
```

An on-time return, `returning_1`, would fill the query identified as `Query_borrowing_1_4` (see footnote 2) which is associated with the obligation instance `being_obliged_function_1(borrowing_1)`. This is detected by the continuous query mechanism, which fires an occurrence, say `filling_1`, of the query being filled (completely). We then derive the following fulfillment occurrence from `fulfilling_function_1`:

```
fulfilling_function_1(filling_1)
        fulfilled:     being_obliged_function_1(borrowing_1)
```

The obligation to return a book before 15th June 2001 is violated when 15th June 2001 passes and the obligation is still in force (has not ceased through fulfillment or being made void). The obligation is also violated when any occurrence occurs that makes its fulfillment impossible, such as destruction of the book. Of course, in the case where the occurrence that makes it impossible to fulfill the obligation is as a result of an 'Act of G-d' (as opposed to act of dog), we may view the obligation as voided, rather than violated, and thereby forgive the non-fulfillment.

The function that describes how violations of John's 1st obligation (`being_obliged_function_1(borrowing_1)`) are brought about is as follows:

```
violating_function_1
        domain:        X = first occurrence, on or before cessation of
                           being_obliged_function_1(borrowing_1),
                           of being_impossible for John to return book_copy_1235
                           before 15th June 2001.
        range:         violating_function_1(X) where
                               violated:
                    being_obliged_function_1(borrowing_1)
```

An occurrence of 15th June expiring without the occurrence of John returning the book implies an occurrence of `being_impossible` that is in the domain of `violating_function_1`. Assume that 15th June 2001 expires and that our system counts the occurrences of John returning the book before 15th June, and finds a result of zero. The monitoring system then asserts `being_impossible_1` which says that occurrences of John returning the book before 15th June 2001 are impossible (since 15th June has passed and none occurred). The occurrence `being_impossible_1` then implies the following violation occurrence:

```
violating_function_1(being_impossible_1)
        violated:      being_obliged_function_1(borrowing_1)
```

What prohibition is this obligation equivalent to? It might be tempting to think of the obligation to return a book before 15th June 2001 as equivalent to the prohibition against returning the book after 15th June 2001. However, we must be careful of our reading and implementation of this prohibition. We do not want a prohibition against *returning* the book after 15th June 2001, since this would fire a violation each time the book is returned after 15th June 2001, which is not what we intend. 'Better a late return than no return' is not captured by that logic interpretation. Rather, our prohibition is against *the single case where* no books are returned before 15th June, and the violation conditions for this have been defined already in `violating_function_1`.

4 Situation

As deontic logics generally neither capture the provenance of a norm (e.g. its author, specification time, or document position), nor a unique identifier for each norm, conflict resolution in formal logic is in many cases untenable as insufficient information about the policies exist to enable choice amongst them. We can rectify this by associating each identified norm (function or specific obligation instance) with a clause that construes that obligation as having come about. This may be achieved by adding an '`isAccordingTo`' role to each function, obligation, permission, or prohibition, where the participant in this role is a clause identifier (such as `Clause_1`) which identifies the utterance that promulgated this clause. Using an occurrence of `being_in`, the utterance identifier can then be associated with a document position identifier (e.g. 'Section 1.1'), and a document heading ('Library General Regulations'). Alternatively, the clause can be associated with an utterer (e.g. 'Chief Librarian') or institution which associates itself with the clause (e.g. 'Library of Congress'), and an utterance time and place (e.g. 'Washington', 'February 8th 2002').

As each obligation is *according to a particular clause*, these obligations may be considered as a representation of the notion of *prima facie* obligations discussed in the deontic logic literature (Prakken and Sergot 1997). John might have a prima facie obligation, according to Clause 1 (in Library General Regulation 32B) to return the book *within 14 days*. But, being a faculty member, John may also have another prima facie obligation, according to Clause 2 (in Library General Regulation 48C) to return the book *within 30 days*. The general norm that brings about the latter prima facie obligation may be represented by the function `being_obliged_function_2`:

```
being_obliged_function_2
        domain:         X = any occurrence of borrowing, where borrower is a
                            faculty member, from library
        range:
                being_obliged_function_2(X) where
                        obliged :       first occurrence, on or after date of X,
                                        and within 30 days,
                                        of borrower in X returning borrowed in X
                                        (to lender in X)
                        isAccordingTo: Clause_2
```

We can similarly add '`isAccordingTo: Clause_1`' to the general obligation defined by `being_obliged_function_1`. We will see how this information is used for conflict resolution purposes in Section 5.2. For the moment though, let us compare our intentions when tagging norms with clause identifiers to Sergot et al's (1986) work on relativising norms.

A mechanism of labeling conclusions with the section of law from which they were derived was employed in Sergot et al (1986) in their analysis of the British Nationality Act. That paper recommends that rules take the form:

```
[proposition] on [date] by Section [section number] if [conditions]
```

For example:

```
x acquires British citizenship on 16 March 1987 by Section 11.1 if …
```

Sergot et al recommend that the section number be recorded because candidates may qualify by any of four different section for citizenship, and the way in which the candidate qualified may have implications for, for instance, the nationality of their children. Their intention in recording section numbers is solely to record the origin of a conclusion in a conflict-free specification so that the specific type of citizenship can be derived, where type of citizenship corresponds to section number under which the citizenship is acquired. Our representation differs in a few important respects. Firstly, our identifiers are system-assigned and the utterances (clauses) are regarded as an immutable part of history once recorded. For us, multiple clauses may make conflicting construals without logical contradiction, though clearly choice as to which construal to uphold must be made during application of the rules by the norm interpreter. Such choice is typically guided by other rules: these are high-level norms or 'principles' (selection principles). In contrast, Sergot et al propose that identified sections be revised to eliminate conflicts. There is no allowance for subjective clauses to conflict without logical contradiction, and conflicts are in fact specifically removed by redrafting. For instance, the addition of Section 11.2 which specifies exclusions to the conditions of Section 11.1, would require the restatement of Section 11.1 as:

```
x acquires British citizenship on 16 March 1987 by Section 11.1 if …
and not [x is prevented by section 11.2 from acquiring British citizenship]
```

Such revisions are problematic from a number of perspectives. Firstly, as Sergot et al point out, if conflict elimination is to be achieved, adding sections requires the adjustment of existing rules (primarily by human trial and error), which becomes increasingly difficult as the size of the rule-base grows. The addition of a single new rule may require the revision of many existing rules, violating the principle of locality of update. Secondly, we believe, adjusting existing rules changes the content of what was proposed by the legislator. Section 11.1 above is no longer the same section as contained in the original Act, since it now contains additional conditions from later sections and even interrelated laws. By repeatedly revising existing rules, a prima facie provision becomes altered to an all-things-considered provision, and we are no longer concluding

```
x acquires British citizenship on 16 March 1987 by Section 11.1
```

but rather, effectively,

```
x acquires British citizenship on 16 March 1987 by the law as a whole
```

The provisions of the section are being confused with the provisions of the law as a whole, thereby diluting the usefulness of the section identifier. Hansson and Makinson (1997) make a very useful distinction between "contraction-and-revision", which is what Sergot et al employ, and "restrained application". The former involves derogation and amendment: adding, removing, and altering rules. The latter leaves the rules exactly as stated, but chooses one rule above another when application of the rules by a norm-interpreter gives contradictory results. Makinson (1998) suggests that, while inconsistency is never entirely eliminable, legislation should be redrafted so far as possible to remove conflict, since pervasive inconsistencies complicate application of the norms. Revision of law to eliminate conflict is useful in many instances, and Sergot et al's recommendations are pertinent to legislators and policy makers. However, revision of the clausal text of existing legislation and policy during implementation to account for newly introduced rules fundamentally alters its meaning and is to be avoided. Restrained application (choosing from a set of conflicting construals on a case-by-case basis) may be required instead. This is not to say that we object to document revision altogether. Our objection is against changing the contents of an utterance once uttered: once a rule has been stated, it is always the case that its particular contents were stated, and the contents of the utterance are therefore immutable, even though the contents of a particular document may be subject to change. The intention of document-level labeling ('Section 11.1', paragraph headings, document headings such as 'British Nationality Act') we see as being useful in allowing the determination of which portion of text a conclusion was derived from; that is, the determination of where a particular utterance was included in a named document or section at a given time. Altering rule text through rule revision makes such determination very difficult indeed, since multiple texts are being coalesced, thus blurring their distinct specifications: Section 11.1 is no longer the same Section 11.1 as the law-makers intended since it contains different utterances from their original statements. We see revision as being the case where a particular utterance is included in, say, 'Section 1.1' at some time, and a *different*, identified utterance is included in Section 1.1 at a later stage. We have no objection to the revision of *law* but we do object to the revision of *history*. The content and positioning of text in the British Nationality Act of 1981 at the time it was passed is inalterable. Subsequent amendments do not change what was passed, since that is history, but may change what is to be considered by the judiciary (or decision-maker) in their application of the law (or policy) to specific cases.

5 Conflict Specification, Detection, and Resolution

In our example above, we saw that obligations are associated with a description of the set of occurrences that can fulfill that obligation. When the query describing that set is filled, the obligation is fulfilled. We also saw how a query may be used to define the domain of functions. In the case of obligation functions, the occurrences produced by the function represent specific, identified obligations arising from occurrences in the domain of the general obligation function. In the case of prohibitions and permissions, we can also use queries to describe the prohibited and permitted occurrences. The function device is used to produce case-specific instances of permissions and prohibitions, from the general permissions and prohibitions.

Assuming a rule that all borrowings are permitted, we might represent this using the following function:

```
permitting_function_1
        domain:        X = any occurrence of borrowing from library
        range:         permitting_function_1(X) where
                               permitted:     X
                               isAccordingTo: Clause_3
```

We can see that, prima facie, `borrowing_1` is permitted, as it falls in the domain of the function, and therefore generates the occurrence:

```
permitting_function_1(borrowing_1)
        permitted:     borrowing_1
        isAccordingTo: Clause_3
```

Assuming a rule that borrowings of rare books are prohibited[4], we might represent this using the following function:

```
prohibiting_function_1
        domain:        X = any occurrence of borrowing a rare book from library
        range:         prohibiting_function_1(X) where
                               prohibited:    X
                               isAccordingTo: Clause_4
```

An additional fact may specify that `Clause_4` is contained in our Book Preservation Regulations. Assuming that `book_copy_1235` is not a rare book we can see that, prima facie, `borrowing_1` is not prohibited according to Clause 4, as it does not fall in the domain of the function. But what if `book_copy_1235` is a rare book? Then it would be clear that, prima facie (according to Clause 4), `borrowing_1` is prohibited, as it falls in the domain of the function and therefore produces the occurrence identified as:

```
prohibiting_function_1(borrowing_1)
        prohibited:    borrowing_1
        isAccordingTo: Clause_4
```

Here we notice a conflict: `borrowing_1` is both permitted and prohibited. Let us deal firstly with mechanisms for analytically determining conflicts in advance (§5.1), and then with conflict resolution mechanisms (§5.2).

5.1 Conflict Detection

Since norms are associated with queries that describe sets of occurrences, conflicts may, in general, be detected by analytically determining overlap between stored queries. This conflict detection may be driven by some basic suggestions of deontic logic[5], which propose that conflicts exist when:

1. an occurrence is in a set of permitted occurrences and a set of prohibited occurrences.
2. an occurrence is in a set of obliged occurrences and a set of prohibited occurrences
3. an occurrence is in a set of obligatory occurrences (implying that not performing the action produces a violation), but a permission to refrain from performing the occurrence exists (implying that no violation arises from not performing the action) (Lee 1988; Makinson 1988)
4. an occurrence is in the domain of a function but is in a set of forbidden occurrences. This principle can be derived from the principle that a power may conflict with a prohibition against exercising that power (Makinson, 1986; Jones and Sergot 1996). This is because, effectively, functions define powers, because functions define what set of occurrences can bring about, according to a certain clause, other occurrences.
5. an occurrence is in a set of obliged occurrences but is not in the range of a function. This can be derived from the principle that an obligation may conflict with the absence of a power to fulfil that obligation (the latter includes the case where another party has immunity against a certain state of affairs being brought about).

From point 1 above we can see that, as the domains of `permitting_function_1` and `prohibiting_function_1` overlap, there is potential for conflict. Unlike Standard Deontic Logic, we see

[4] Take 'prohibited' here to mean 'a violation results if it occurs', rather than "no person is legally empowered to bring about the state of affairs where the library views the book as 'borrowed'".

[5] The authors cited here do not make use of a notion of occurrences, but reason at the level of propositions and contradiction between propositions. We restate their suggestions in terms of sets of occurrences.

no logical contradiction in the co-existence of conflicting prima facie permissions and prohibitions. Instead, we argue that, where dilemmas exist, some choice must be made as to which particular permission or prohibition entity to *void* or *violate* in a given case. Conflict resolution is dealt with next.

5.2 Conflict Resolution

A variety of conflict resolution options are available to us:

- revision to eliminate conflict. This approach is taken by Sergot et al. (1986).
- amendment to specify which case-specific instances are *void* for a particular case
- in the absence of revision or amendment, acceptance that fulfilling one provision may *violate* others.

In the presence of conflicting provisions, we may wish it to be the case that one or more of the provisions is *voided*, or we may wish simply to guide the decision-maker as to which provisions to *violate*. While we agree that revisions to the rule-set may be useful in removing conflict, we see revision alone as insufficient. Supplementation of the rule-set with additional choice principles for case-based reasoning is a necessary and complementary conflict resolution mechanism: here we specify which case-specific obligation, permission, or prohibition instances are *voided*. Supplementation of the rule-set with choice principles for deciding which norms to *violate* in cases where violations are unavoidable is also possible.

In the above example, given that a borrowing may fall into both the domain of `permitting_function_1` and the domain of `prohibiting_function_1` we can define a function, `voiding_function_4`, that specifies which generated permission or prohibition instance is void in the circumstance. Let us assume that our normative system says that the specific permission to borrow a book is voided where prohibitions defined in our Book Preservation Regulations contradict. `voiding_function_4` is a general norm which specifies which provision's construal (i.e. instantiation of a norm) is void in the light of conflicting prima facie instances.

```
voiding_function_4
        domain:         X = occurrences produced by permitting_function_1 from
                            occurrence Y, where Y is in the role 'prohibited'
                            in any occurrence of prohibiting where the
                            isAccordingTo role is a clause in
                            Book Preservation Regulations
        range:          voiding_function_4(X) where
                                voided:        X
                                isAccordingTo: Clause_5
```

In the presence of `borrowing_1`, this function produces the following result:

```
voiding_function_4(permitting_function_1(borrowing_1))
        voided:         permitting_function_1(borrowing_1)
        isAccordingTo:  Clause_5
```

It is then clear that, according to `Clause_5`, the permission identified as `permitting_function_1(borrowing_1)` is void in this circumstance. It should be noted here that it is the *case-specific instantiation* (`permitting_function_1(borrowing_1)`) of the permission that is voided, rather than the *permission in general* (`permitting_function_1`). Hence, the general permission may still apply to other occurrences of borrowing. The distinction between case-specific permission instances and the general permission from which they are derived is at the heart of Hansson and Makinson's (1997) contrast between restrained application and revision. In restrained application (typically used by judges in their application of the law), it is the case-specific permission instance that is voided. In revision (typically used by legislators in their revision of the law), it is the general permission that is voided.

It may be argued at this stage that if we ask the question 'is `borrowing_1` permitted?' we get the response 'yes', since `permitting_function_1(borrowing_1)` is stored in our database. However, the 'yes' is actually a qualified 'yes': what we really mean is 'prima facie, according to `Clause_3`, yes'. But looking further we see that `permitting_function_1(borrowing_1)` is voided. What we need is an additional definition of permission that says that a void permission is, in some sense, not a permission at all. We may define:

```
permitting_function_2
        domain:         X = any occurrence of permitting but not one that
                            participates, in role 'voided' in an occurrence
                            of voiding
        range:          permitting_function_2(X) where
                                permitted:      |X|.permitted
                                isAccordingTo: Clause_6
```

Clearly `permitting_function_1(borrowing_1)` is not in the domain of `permitting_function_2` since it participates in the role `voided` in `voiding_function_4(permitting_function_1(borrowing_1))`. Once we have this definition of what it means for something to currently be permitted, we can ask the specific question 'is `borrowing_1` permitted according to `Clause_6`?' our answer is an unambiguous 'No'.

As another example of conflicting prima facie norms, consider the case where John is a faculty member. Under `being_obliged_function_1` and `being_obliged_function_2` defined earlier (§2.2 and §4 respectively), we have, arising from `borrowing_1`, the following two obligations:

- `being_obliged_function_1(borrowing_1)` saying John is obliged to return the book by 15 June
- `being_obliged_function_2(borrowing_1)` saying John is obliged to return the book by 31 June

At first glance, it appears that there is no conflict here: John could return the book before 15th June 2001 and so satisfy both obligations. However, in this case, let us assume that faculty members are exempt from any obligations produced by `being_obliged_function_1`. We can proceed as we did for the voiding of the case-specific prohibition above: by defining a function that specifies which specific instances are void.

```
voiding_function_5
        domain:         X = occurrences produced by being_obliged_function_1 from
                            occurrence Y, where Y is a borrowing where the
                            participant in the role 'borrower' is a faculty member
        range:          voiding_function_5(X) where
                                voided:         X
                                isAccordingTo: Clause_7
```

In the presence of `borrowing_1`, this function produces the following result:

```
voiding_function_5(being_obliged_function_1(borrowing_1))
        voided:         being_obliged_function_1(borrowing_1)
        isAccordingTo: Clause_7
```

So, the obligation to return the book by 15th June is here void.

Voiding obligations produced by conflicting clauses is one possible solution to conflict; another, is to merely accept that conflict exists. In many environments it may be the case that the stricter obligation is intended to be enforced even when the more lenient obligation also applies. This is hard to imagine when the obligations both arise from a single source, but consider the case where library regulations say that returns by 31st June are required, whereas the principles of general fairness say John should nevertheless return the book by 15th June. Here we cannot easily argue that one obligation voids the other; rather they exist in conflict, and John must choose which to violate. If John returns the book on 17th June there is then one violation: a violation of a principle of general fairness. If John never returns the book, he has violated both obligations: there is a violation of a principle of general fairness and a violation of a specific library regulation.

6 Time

In the spirit of Sergot (1986), we tag each conclusion with a time to indicate the moment at which the conclusion was derived. A conclusion derived at time t1, may no longer be derived at time t2. Nonmonotonicity (revision of conclusions) is supported, as partial information is supplemented over time. A continuous query mechanism, which monitors how the results of stored queries change over time still remains applicable here. Occurrences join and leave the domain of functions over time; consequently, the range of functions changes over time. Where necessary, we can tag the item produced by the function with the time of its production.

Let us define a high-level principle that says that something is (currently) obliged if there is an obligation to do it which has not yet ceased[6]:

```
being_obliged_function_42
        domain:         X = any occurrence of being_obliged
                            that does not participate in role 'ceased' in an
                            occurrence of ceasing.
        range:          being_obliged_function_42(X) where
                                obliged:        |X|.obliged
                                isAccordingTo: Clause_42
```

We have called this principle `Clause_42`. Now, beginning with an empty database, let us trace our book-borrowing scenario from its start until just after the first return of the book. Below is a commented transcript: `%` represents a command prompt with user input, `>` represents a system response to a query, and `=>` represents an output from system derivation. For brevity, role-players in each identified occurrence are not indicated; the reader may consult the respective occurrence identifiers mentioned earlier in this document for further details on the participants in each identified occurrence and their roles.

Date	Commentary	Session Transcript
1 Feb 2001	Borrowers are obliged to return books borrowed (`Clause_1`)	`% INSERT being_obliged_function_1`
	Is John obliged to return the book, according to `Clause_42` ?	`% SELECT occurrences of being_obliged WHERE isAccordingTo=Clause_42 AND obliged >= Query1`
	No, he's not. (He hasn't yet borrowed it.)	`> 0 results.`
1 June 2001	John borrows the book	`% INSERT borrowing_1`
	Therefore, prima facie, according to `Clause_1`, John is obliged to return the book.	`=> being_obliged_function_1(borrowing_1)`
	Also, prima facie, according to `Clause_42`, John is obliged to return the book.	`=> being_obliged_function_42( being_obliged_function_1(borrowing_1))`
	Is John obliged to return the book, according to `Clause_42` ?	(Same `SELECT` query as above)
	Yes, he is.	`> being_obliged_function_42( being_obliged_function_1(borrowing_1))`
3 June 2001	John returns the book.	`% INSERT returning_1`
	Continuous query mechanism detects query associated with obligation is filled	`CQ monitor: INSERT filling_1`
	Therefore, the obligation to return the book has been fulfilled.	`=> fulfilling_function_1(filling_1)`
	And, the obligation to return the book ceases.	`=> ceasing_function_1( fulfilling_function_1(filling_1))`
	Is John obliged to return the book, according to `Clause_42` ?	(Same `SELECT` query as above)
	No, he is not. (According to `Clause_42`, something is no longer obliged if the obligation to do it has ceased.)	`> 0 results.`

From the transcript above, we can see that the question 'is John obliged to return the book, according to `Clause_42` ?' is answered 'No' on 1st Jan 2001, 'Yes' after his borrowing on 1st June 2001, and 'No' again after his book return on 3rd June 2001. Conclusions are defeasible. On 5th June 2001, we could ask whether it was the case, on 2nd June 2001 at 14h00, that John was obliged to return the book. We could do this by running the `SELECT` query from the transcript above, but instructing the inference engine only to use facts and rules dated on or before 2nd June 2001 at 14h00. The query, framed on 5th June but using only information known up until 2nd June 2001 at 14h00, would return `being_obliged_function_42(being_obliged_function_1(borrowing_1))` meaning that, yes, as at 2nd June 2001 at 14h00, John was obliged to return the book.

[6] We defined what it means for an obligation to 'cease' through a `ceasing_function` in §3.

7 Related Work

We have presented problems raised by: (1) the relationship between *obligations and time*, and (2) the *origin of obligations and conflicts*.

In real problems, many obligation statements refer to time, and obligation validity changes over time. The relationship between *deontic notions and time* has previously received some attention. For instance, Brown (1996, 2000) explains that dynamic, dischargeable obligations (e.g. 'to return the book'), are more prevalent in contracts than the static, standing obligations (e.g. 'to honor your parents') of SDL. However, the temporal logic Brown defines does not allow one to date obligations, and the problem of life cycle of individual, identified obligations over time is not addressed.

In terms of problems raised by the *origin of obligations and conflicts*, a previous contribution to the literature on defeasible deontic logic and non-monotonic reasoning with contradictory norms in this regard is the work by Cholvy and Cuppens. They speak of merging sets of norms originating from conflicting roles (Cholvy and Cuppens, 1995) or regulations (Cholvy and Cuppens, 1998). In their earlier work, sets of obligations, permissions, and prohibitions are associated with identified roles – such as 'Role1=Christian' or 'Role2=Ordered_Soldier'. In the latter work, sets of norms are associated with identified regulations – such as 'Regulation1=Eating Regulations' and 'Regulation2=Asparagus Eating Regulations'. Cholvy and Cuppens (1998) extend SDL by relativizing norms to regulations, expressing sentences of the form '*within regulation R / regulation R says / according to regulation R*, it is obligatory / permitted / forbidden that P'. A precedence ordering relationship is defined between regulations. When regulations are merged, conflicts in the norm sets are resolved by allowing norms from the regulation with higher precedence to override. The notion of precedence-based override is seen as a generalization of Horty's suggestion of specificity-based override, as specificity ordering is viewed as one means of preference ordering. Taking the rules "generally, one ought not to eat with one's fingers" and "when eating asparagus, one ought to eat with fingers", specificity-based override takes the second rule over the first one, since its premise is more specific. Cholvy and Cuppens' approach is reasonably course-grained as all norms in regulation R1 are taken to override all norms in regulation R2. The logic does not speak of dynamic determination of context-specific priorities: it does not explicitly attend to the problem that precedence relationships are non-static and in some cases regulation R1 overrides R2, whereas in other circumstances, R2 overrides R1. Cholvy and Cuppens assume that regulations are internally consistent and conflict free, and their logic therefore does not attend to conflicting obligations brought about by different events covered by the same norm. For example, Hansson and Makinson (1997) give the case where the single rule that 'the doctor must immediately visit heart attack victims' generates conflicting obligations when two remote patients suffer heart-attacks: an obligation to visit patient A immediately and a separate, conflicting one to visit patient B immediately. Cholvy and Cuppens treatment of the *source or origin of obligations* (and conflicts) effectively looks at their *provenance in documents* but not at their *birth in circumstance*.

8 Conclusion

We have critically reviewed the traditional operator-based construal of obligation and permission in deontic logic and illustrated the usefulness of *identifying* specific obligation, permission, and prohibition instances which are generated from norms by functions on occurrences. We have shown that the identification of separate case-specific instantiations of obligations (similarly permissions and prohibitions) allows us to speak of the *life-cycle* of those obligations, and to reason about their force over *time* as new information becomes available. We explained how the supplementation of deontic specifications with additional information to capture the *situation* or provenance of a norm (e.g. author, document position, place of utterance, etc.) allows us to reason more powerfully about conflicts between prima facie provisions. Through worked examples, we demonstrated the process of reasoning in the light of conflicting norms, which may be voided or violated. We believe our exposition enlightens aspects of the life and times of identified, situated, and conflicting instantiations of norms that are not dealt with in standard operator-based treatments of norms in deontic logic.

9 Acknowledgements

The authors would like to acknowledge the valuable feedback received from the anonymous referees, helpful input kindly provided by David Makinson, and the hospitality of the Information and Decision Technologies Group at the Department of Operations and Information Management, Wharton Business School, University of Pennsylvania, which hosted one of the authors during the writing up of this work. This research is supported by grants from the Cambridge Commonwealth Trust, the Overseas Research Students Scheme (UK), and the University of Cape Town Postgraduate Scholarships Office.

10 References

Abrahams, Alan S., and Jean M. Bacon. (2001). 'Representing and Enforcing E-Commerce Contracts Using Occurrences', *Proceedings of the Fourth International Conference on Electronic Commerce Research*, Volume 1, Dallas, TX: ATSMA, IFIP, INFORMS, 59-82.

Allen, James. (1995). *Natural Language Understanding*, 2nd ed. Menlo Park, CA: Benjamin Cummings, 1-41, 248.

Anderson Alan R. (1958). 'A reduction of deontic logic to alethic modal logic', *Mind* 67. 100-103.

Bennett, Jonathan. (1988). *Events and their Names*. Oxford, UK: Oxford University Press.

Brown MA. (1996). Doing As We Ought: Towards a Logic of Simply Dischargeable Obligations. In Brown MA and Carmo J (eds): *Deontic Logic, Agency, and Normative Systems: Proceedings of the Third International Workshop on Deontic Logic In Computer Science (DEON'96)*. Sesimbra, Portugal. Springer. January 1996.

Brown MA. (2000). Conditional Obligation and Positive Permission for Agents in Time. *Nordic Journal of Philosophy* 5, no. 2, 83-112

Cholvy L and Cuppens F. Solving normative conflicts by merging roles. *In Proceedings of the Fifth International Conference on Artificial Intelligence and Law (ICAIL'95)*. Washington, USA. pp 201-209. 1995.

Cholvy L and Cuppens F. Reasoning about norms provided by conflicting regulations. In Prakken H and McNamara P (eds): *Norms, Logic, and Information Systems: New Studies in Deontic Logic and Computer Science*. IOS Press, Amsterdam. pp. 247-264. 1998.

Daskalopulu, Aspassia and Marek J. Sergot. (2001). 'Computational Aspects of the FLBC Framework', *Decision Support Systems*. Special Issue on Formal Modeling in E-Commerce.

Davidson, Donald. (1980). *Essays on Actions and Events*. Oxford, UK: Clarendon Press.

Hansson, Sven O., and David C. Makinson. (1997). 'Applying Normative Rules with Restraint', in ML Dalla Chiara et al (eds.), *Logic and Scientific Methods*. Dordrecht: Kluwer, 313-332.

Jones, Andrew J.I., and Marek J. Sergot. (1993). 'On the Characterization of Law and Computer Systems: The Normative Systems Perspective', in Meyer John-Jules Ch. and Roel J. Wieringa (eds.), *Deontic Logic in Computer Science: Normative System Specification*. Chichester, UK: John Wiley & Sons.

Jones, Andrew J.I. and Marek J. Sergot. (1996). 'A formal characterisation of institutionalised power', *Journal of the Interest Group in Pure and Applied Logic* 4, no. 3, 427-443.

Kimbrough, Steven O. (1998). 'On ESΘ Theory and the Logic of the X12 Date/Time Qualifiers', *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS98)*. Kohalo Coast, Hawaii: IEEE Computer Society Press.

Kimbrough, Steven O. (2001). 'Reasoning about the Objects of Attitudes and Operators: Towards a Disquotation Theory for the Representation of Propositional Content', *Eighth International Conference on Artificial Intelligence and the Law (ICAIL 2001)*. St Louis, Missouri: ACM Press.

Kowalski Robert and Marek Sergot. (1986). 'A logic-based calculus of events', *New Generation Computing* 4, 67-95.

Lee, Ronald M. (1988). 'Bureaucracies as Deontic Systems', *ACM Transactions on Office Information Systems* 6, no. 2 (April), 87-108.

Makinson, David. (1986). 'On the Formal Representation of Rights Relations', *Journal of Philosophical Logic* 15, 403-425.

Makinson, David. (1988). 'Rights of Peoples: Point of View of a Logician', in James Crawford (ed.), *The Rights of Peoples*. Oxford, UK: Oxford University Press, 69-92.

Makinson DC. (1999). On a Fundamental Problem of Deontic Logic. In McNamara P and Prakken H (eds.): *Norms, Logic, and Information Systems: New Studies in Deontic Logic and Computer Science*. Amsterdam: IOS Press. 29-53.

Parsons, Terence. (1990). *Events in the Semantics of English: A Study in Subatomic Semantics*. Cambridge, MA: MIT Press.

Pianesi, Fabio and Achille C. Varzi. (2000). 'Events and Event Talk: An Introduction', in James Higginbotham, Fabio Pianesi, and Achille C. Varzi (eds.), *Speaking of Events*. Oxford, UK: Oxford University Press, 3-49.

Prakken, Henry and Marek Sergot. (1997). 'Dyadic Deontic Logic and Contrary-to-Duty Obligations', in Donald Nute (ed.), *Defeasible Deontic Logic: Essays in Nonmonotonic Normative Reasoning*, Synthese Library No. 263, Kluwer Academic Publishers, 223-262.

Sergot, Marek J., et al. (1986). 'The British Nationality Act as a Logic Program', *Communications of the ACM* 29, no. 5 (May), 370-386.

Sowa, John F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks/Cole.

11 Appendix: Supplementary Example

Let us consider an additional example which treats most of the issues discussed in this paper. In Jewish tradition, a Cohen is forbidden from visiting a cemetery, but may attend the funeral, which necessarily involves visiting a cemetery, if their parent or child dies. General principles of good faith say a person is obliged to attend the funeral of family members and close friends. We wish to illustrate in this section two different treatments of conflicting norms across two different cases:

1. in the first case, a visit for a parent's funeral, a case-specific instance of a conflicting norm is *voided*
2. in the second case, a visit for a friend's funeral, a case-specific instance of a conflicting norm is *violated*

The examples demonstrates that, in some cases, it is important to resolve conflicts by designating norms as void in a circumstance: that is, by designating case-specific instances of those norms as void. However, in other case, merely detecting violating of conflicting norms is sufficient.

Let us examine the general norms in place, and their case-based instantiations.

We have a general prohibition against visits of Cohens to cemeteries which gives rise, for instance, to a specific instantiation of the prohibition, being the prima facie prohibition against a visit, say a visit by Jeff Cohen to West Park Cemetery for his father's funeral. The specific prohibition is generated from the general prohibition by a function: the function takes as its domain the set of prohibited occurrences. Take the case where Jeff Cohen's father, Jack, has died, and Jeff visited West Park Cemetery on 1st May 2003 for his father's funeral. Assume we wish to now assess, retrospectively, whether Jeff's action was permissible. Take Jeff's visit as `visiting1`. Assuming the general prohibition against Cohen's visiting cemeteries was identified as `prohibiting_function_6`, which takes as its domain any visit by a Cohen, we have the prima facie prohibition, identified as '`prohibiting_function_6(visiting1)`' against `visiting1`. We derive that, prima facie, `visiting1` is prohibited.

We also have a function, call it `being_obliged_function_7`, which takes as its domain the set of occurrence of deaths, and for each, creates a specific obligation instance upon persons who are close family or friends of the deceased to attend the funeral. Taking Jack Cohen's unfortunate death as `dying1`, we then have the specific obligation, `being_obliged_function_7(dying1)`, upon his son Jeff to visit West Park Cemetery.

Taking `visiting1` as the visit that fulfilled this obligation, we then have a conflict: `visiting1` was, prima facie, both obliged and prohibited. Given, though, that the intention of the law is that the obligation, `being_obliged_function_7(dying1)`, should defeat the prohibition as the visit is for the death of a parent, we can mark `prohibiting_function_6(visiting1)` as void, through its participation in role `voided` in an occurrence `voiding_function_8(visiting1)`, thereby eliminating the conflict. We conclude that `visiting1` was not prohibited since `prohibiting_function_6(visiting1)` is void (i.e. not regarded as a valid prohibition by a high-level principle of law which says that void prohibitions do not count as prohibitions).

Now, assume that the unfortunate Jeff Cohen happens to lose his best friend Marvin Goldberg (who is no relation to Jeff). Take Marvin's death as `dying2`. By general principles of good faith, Jeff has the prima facie obligation, `being_obliged_function_7(dying2)`, to attend Marvin's funeral. Assume Jeff, disrespectful of Jewish law, visits the cemetery to attend Marvin's funeral. Take this visit as `visiting2`. `visiting2` fulfils `being_obliged_function_7(dying2)`. Now `visiting2` is clearly not for the death of a parent or child, and so `prohibiting_function_6(visiting2)` is in force. Seeing as the prohibition is in force, `visiting2` brings about a violation, `violating_function_8(visiting2)` where it is `prohibiting_function_6(visiting2)` that is violated. What arises here is a choice as to which conflicting norm instance to violate in this case: Jeff chose to violate Jewish law, rather than general principles of good faith. Had he stayed home, he would have violated general principles of good faith, but obeyed Jewish law. The choice as to which to violate was in this case forced on him, as he could not physically have honored his obligation and respected the prohibition.

Note that a prohibition that is void *in a particular case* is not void *in general*: so voiding a particular *case-based instantiation* of a prohibition is not the same as voiding the *general* prohibition. Voiding the prohibition against Jeff Cohen visiting West Park to attend his father's funeral on 1st May does not void the prohibition against his other visits in other circumstances: that is, he is still prohibited from visiting the next day, or even from visiting the same day but for a different purpose. The English word 'prohibition' is loosely used to refer to both general prohibitions and particular case-based instantiations of general prohibitions, but it is important for reasoning purposes to separately identify the notions so that case-based instantiations can be voided without voiding the general prohibition. Similarly, for obligations, identifying case-based instantiations is essential. Note that it is not the general prohibition, `prohibiting_function_6(X)`, against Cohens visiting cemeteries that is regarded as null, since that prohibition may still apply to other Cohen's who were not relatives of the deceased, but the *specific case-based instantiation* of the prohibition - i.e. the prohibition, `prohibiting_function_6(visiting1)` (generated from the general prohibition), against Jeff Cohen visiting West Park Cemetery - that is regarded as null. Again, we see the distinction between revision and restrained application illuminated by Hansson and Makinson (1997): revision would have involved the nullification of the general prohibition (and hence all case-based instances), whereas restrained application involves the nullification of a specific case-based instance only. As we have argued previously, *nullification* (voiding) of a case-based instance is only one means of dealing with conflict though: the alternative is to simply recognize that (in the case of `visiting2`) the prohibition is *violated*.

In SDL, Jeff's obligation to visit the cemetery for his friend's funeral, and the prohibition against him visiting the funeral would yield a contradiction:

$$O\alpha \wedge \neg P\alpha \rightarrow \bot$$

In contrast, in cases where neither the obligation nor the prohibition is voided, we see it that the following would be the case:

$$\exists e_1, e_2\ \text{being_obliged}(e_1) \wedge \text{obliged}(e_1, \alpha) \wedge \text{prohibiting}(e_2) \wedge \text{prohibited}(e_2, \alpha) \rightarrow$$
$$(\neg(\exists e_3\ \text{ceasing}(e_3) \wedge \text{ceased}(e_3, e_1))) \vee$$
$$(\exists e_4\ \text{violating}(e_4) \wedge \text{violated}(e_4, e_1)) \vee$$
$$(\exists e_5\ \text{violating}(e_5) \wedge \text{violated}(e_5, e_2))$$

That is, if an obligation (e_1) to do α exists, and a prohibition (e_2) against doing α exists, then it is either the case that the obligation has not yet ceased, or it is the case that the obligation was violated, or it is the case that the prohibition was violated.