A Specification Language for Static, Dynamic and Deontic Integrity Constraints

John-Jules Meyer
Hans Weigand
Roel Wieringa

Department of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands

uucp addresses: jules@cs.vu.nl, hansw@cs.vu.nl, roelw@cs.vu.nl

ABSTRACT

In the proof-theoretic view of knowledge bases (KB's), a KB is a set of facts (atomic sentences) and integrity constraints (IC's). An IC is then a sentence which must at least be consistent with the other sentences in the KB. This view obliterates the distinction between, for example, the constraint that *age* is a non-negative integer (which is true of the universe of discourse (UoD) but may be false in a particular implementation of a KB), and the constraint that a class must have precisely one teacher (which is false of the UoD if a class actually has two teachers). The second constraint is called *deontic* and constrains the UoD; the first constraint is a *necessary* truth of the UoD and does not constraint the UoD. Instead, it constrains the implementation of the KB. We show that both types of constraints can be specified in the single framework provided by a deontic variant of dynamic logic, which has the added advantage of being able to specify dynamic constraints as well. We give a non-trivial example of a KB specification with static, dynamic and deontic constraints.

Subject areas: Deductive databases and knowledge-based systems, logical fundamentals of database theory.

1. Introduction

Over the past ten years, there has been active research in the specification of static and dynamic integrity constraints (IC's), the verification of their internal consistency, and the validation that a particular KB satisfies the constraints. (For specification see Dignum et al. [1987], Lipeck [1986], Nicolas & Yazdanian [1978], Reiter [1984], Reiter [1988], Sernadas [1980], Wieringa & van de Riet [1988], for verification see Kung [1985]) and for validation see Ehrich et al. [1984], Fiadeiro & Sernadas [1988], Lipeck & Saake [1987], Lloyd et al. [1987], Nicolas [1982], Weber et al. [1983].) A sharp distinction has been made in this research between *static* IC's, which limit the set of allowable states of a KB, and *dynamic* IC's, which limit the allowable behavior of the KB. An example of a static IC is the statement that the age of a person is always a non-negative number, and an example of a dynamic constraint is the statement that a library user must return a book within six weeks after borrowing it.

Less clarity exists about the difference between IC's which constrain the allowable states or behavior of an implemented KB and IC's which constrain the allowable states or behavior of the universe of discourse (UoD). For example, $age \in \mathbb{N}$ is a truth about the UoD which does not constrain the possible states of the UoD, because it follows from the meaning of the word "age". However, it does constrain the allowable states of a KB, because it says that the KB must not have negative *age* fields. It is precisely because we know that a person cannot have a negative age (assuming the usual meaning of the word "age") that we know that a KB is in an erroneous state when an *age* field is negative.

On the other hand, the rule that a library user ought to return a book within six weeks constrains the behavior of the UoD. If a KB truthfully represents the fact that a user has not returned a book within six weeks, it is not the KB but the user who is in an illegal state. Let us call statements which are true of the UoD and do not constrain the allowable states or behavior of the UoD *necessary IC's* and statements which do constrain the allowable states or behavior of the UoD *deontic IC's* (δεοντως (Greek) = "as it should be, duly"). Combining this distinction with that between static and dynamic IC's we get four possibilities (table 1).

		static	dynamic
necessary	analytical	$age \in \mathbb{N}$	An employee must be hired before s/he can be fired.
	empirical	$age < 150$	All students follow C101 before they do A105.
deontic		The balance of a bank account should not be less than n.	A library user should return a borrowed book after at most 6 weeks.

Table 1

We have subclassified necessary constraints into analytical and empirical IC's. An *analytical* IC is a statement about the UoD whose truth follows from the meaning of the terms occurring in it. An *empirical* IC is a statement about the UoD whose truth must be empirically verified. All analytical IC's are necessarily true, but empirical IC's are not necessarily true. For example, a state of the world can be imagined in which people grow over 150 years of age, which contradicts the empirical statement that $age < 150$. Empirical IC's in general cannot be used to constrain the UoD (people are allowed to become over 150 years of age) nor to constrain the KB (if an *age* field has value 151, it may well be because there is a person of 151 years of age in the UoD). However, for modeling purposes we may classify an empirical truth as a necessary truth when during the period of use of the KB the statement is true, or can reasonably be expected to be true. So, we do not treat $age < 150$ as a necessary truth if we expect to have to represent people of age 150. On the other hand, if we expect no one to become 150 years of age in the near future, we can treat $age < 150$ as a necessary truth. Under this provision, if an *age* field in a KB has value 151, the KB is in an erroneous state. Note, incidentally, that in table 1 "All students follow C101 before they do A105" is deontic if it is a rule which students must obey. In the example we have assumed it is not a rule but an empirical statement about the behavior of students.

The distinction between necessary and deontic IC's is important, because the action on discovering an violation of the IC is different. If a necessary IC is violated, the KB must be corrected, for example by the database administrator. If a deontic constraint is violated, a KB must be able to truthfully represent this. The actions to be taken should occur in the UoD and usually concern the correction of a violation (e.g. a library user who did not return a book must return it). (If the KB can be connected mechanically to objects in a UoD, as in process control systems, the corrective action can be enforce mechanically by the KB.)

Yet, the distinction between necessary and deontic IC's is blurred in much of the literature. Examples of deontic IC's treated as necessary truths are

IC1 "Each car must be registered in the year of its production or in the following year" (Lipeck [1986]),

IC2 "No company must supply two different departments with item I" (Nicolas [1982]),

IC3 "Back orders should be processed at the prices valid when they were first received" (Sernadas [1980]),

IC4 "Everybody, whose qualification is not less than 50, has to earn more than $20,000 per year" (Weber et al. [1983]).

It is, of course, possible to treat these constraints as necessary truths and let the KB constrain the UOD in an absolute way. But such a decision should be made consciously by the appropriate people in the UoD and in the presence of an alternative. In this paper we present such an alternative in the form of a deontic variant of dynamic logic (Meyer [1987], [1988], [to appear]). The language we use to express deontic IC's is called L_{Deon} and contains symbols for predicates (with an initial uppercase letter) and symbols for actions (with an initial lowercase letter). Actions are either permitted or forbidden and may also be obliged, meaning that it is forbidden not to perform the action. The four IC's just mentioned can be expressed as deontic constraints in L_{Deon} as

IC1 $\forall x [Car(x) \Rightarrow [produce(x)] O(register(x)_{(\leq 2)})]$,

"If x is a car, then after it is produced there is an obligation to register it within 2 time units."

IC2 $\forall c, d_1, d_2 [Company(c) \wedge Dept(d_1) \wedge Dept(d_2) \wedge d_1 \neq d_2 \wedge Perf{:}supply(c, d_1, I) \Rightarrow F(supply(c, d_2, I))]$,

"if company c is supplying I to department d_1, it is forbidden for c to supply I to a different department d_2."

IC3 $\forall p, q, d, m, c\ [Product(p) \wedge Number(q) \wedge Date(d) \wedge Number(m) \wedge Customer(c) \wedge$

$Order(p,q,d,c) \wedge Price(p,d,m) \Rightarrow [O(sell(p,q,c,m))]$],
"if p had price m at date d andthere is an order of q items p by c on date d, then there is an obligation to sell q items of p to c for price m."

IC4 $\forall e, q, s, [Emp(e) \wedge Qualification(e, q) \wedge q \geq 50 \wedge Salary(s) \wedge s \leq 20{,}000 \Rightarrow F(init\text{-}salary(e,s)) \wedge F(change\text{-}salary(e,s))]$,
"if e has a qualification of 50 or more, it is forbidden to initialize or change e's salary to less than 20,000."

The rest of this paper has the following structure. In section 2 we define more precisely what we mean by KB's, IC's and IC satisfaction and validation. In section 3 we introduce a simple language for static constraints and define a Kripke structure to interpret this language in. In section 4 we extend this language with dynamic constructs and in section 5 we show how to formulate deontic constraints in this language. In section 5 we sum up the results, compare our approach to deontic constraints to other approaches and list some topics for future research. The appendix contains a non-trivial example of a KB with dynamic and deontic constraints.

2. A model-theoretic view of KB's, IC's and IC satisfaction

We view a KB as an abstract *conceptual model* of the UoD, to be distinguished from the UoD on the one hand and the *data model* of an implementation on the other. A KB is an abstract representation of actual and possible facts in the UoD, of the way these facts may change, and of the way these facts ought to change. If one wishes, the representation of actual and possible facts can be regarded as an abstract database, the representation of the way these facts may change as an abstract knowledge base, and the representation of the way these facts ought to change as an abstract law for the UoD. A data model, on the other hand, is a representation of these facts, knowledge, and norms in a finite machine. Since a KB is an abstract conceptual model, it may be infinite and in this paper we will define it as an infinite set of possible states. A data model, on the other hand, is always finite.

A KB is not only a model of the UoD, it is also a *structure* into which sentences in a formal language can be interpreted. We view any sentence true of the KB as an IC, and if we correctly specified the necessary and deontic constraints obtaining in the UoD as a set T of sentences, then the KB is a model of T in the standard logical sense. An IC is thus a *sentence* which is true of the KB. A theory of the KB is a specification of knowledge in the form of necessary IC's and of norms in the form of deontic IC's, but in our view does not contain any specification of actual or possible individual facts. This contrasts with the usual proof-theoretic view of KB's (Reiter [1984]). Our view has the advantage of considerably simplifying the treatment of predicate completion, as will be explained below.

Turning now to IC satisfaction, there are at least two views of satisfaction of an integrity constraint by a KB (Reiter [1988]). Continuing to view a KB as a model of a set of sentences, according to the *consistency view* of IC satisfaction, KB satisfies IC iff KB can be extended to a model of IC. In the *entailment view*, on the other hand, KB satisfies IC iff it is a model of IC[1]. As Reiter notes, the problem with the consistency view is that in this view the constraint

$$\forall x[Emp(x) \Rightarrow \exists y[ss\#(x,y)]] \qquad \text{IC0}$$

would be satisfied by a KB where the predicate Emp has extension *Mary* and the extension of $ss\#$ is

1. Viewing KB as a set of sentences, the consistency view is that $KB \cup IC$ is satisfiable and the entailment view is that any model of KB must be a model of IC.

empty. This model can be extended to a model of IC0 by adding an unnamed object o such that the tuple $(Mary, o)$ is in the extension of *ss#*. However, like Reiter [1984], we do not accept unnamed objects.

The problem with the entailment view is that the empty model is a model of IC0, for trivially all employee objects in that model have a social security number. In our model-theoretic view of KB's it is natural to define IC satisfaction as satisfaction (in the standard logical sense) of a formula by a structure. We eliminate the problem of empty models by requiring the KB to be a non-empty model of IC.

Definition 1.

For a given logical language L with a class $\mathcal{L}$ of intended models, let IC be (the conjunction of) a collection of integrity constraints expressed in L.

1. The *constraint satisfiability problem* is the problem of checking whether IC is satisfiable, i.e. whether there is a non-empty model $\mathcal{M} \in \mathcal{L}$ such that $\mathcal{M} \models IC$.
2. The *constraint validation problem* for a KB $\mathcal{M} \in \mathcal{L}$ is the problem of checking whether $\mathcal{M} \models IC$. □

Thus the constraint satisfiability problem is a general problem about the *internal* consistency of a set of constraints, whereas the validation problem concerns the issue whether a *particular* KB is a model of IC. The problems exist for analytical, empirical and deontic constraints and for each of these, for static as well as dynamic constraints. It depends on the choice of language L which of these types of constraints can be expressed.

The constraint satisfiability problem is usually called the constraint *verification* problem in the literature. We do not use this term because, first, there already is a perfectly acceptable term from logic and, second, in our view verification should be construed as the problem of checking whether the KB is an accurate abstraction of the UoD, just as in natural science verification is testing a model against a UoD.

Note that with respect to an implementation of a KB in a finite machine, we may want to take the consistency view of IC satisfaction and view some IC's as *derivation rules* for the implementation. A derivation rule is used to derive new information from stored information and can therefore be used to save on storage (Nicolas & Gallaire [1978], Nicolas & Yazdanian [1978]). Because our abstract model is potentially infinite but any implementation is finite, we require of an implementation merely that it can be extended to a model of the theory. The implementation must be consistent with the IC's but need not (and often cannot) itself be a model of the IC's.

3. Static necessary constraints

Static necessary constraints can be expressed in the usual way in any first order language. In order to prepare for the dynamic and deontic extensions later on, we now fix a language L_{Stat} by giving its syntax, a semantics, proof rules and axioms.

Syntax

We do not actually give the variables of L_{Stat} but use the letters x, y and z (possibly indexed) as metavariables over the variables. Constants are $A101$, 1234, ... and the letter c (possibly indexed) is used as metavariable over the constants. There are infinitely many variables and constants. There are finitely many transparent function symbols (see below), with metavariables f, g, ... *Supplier*, *Emp*, ... are predicate symbols and the letters P, Q, R are used as metavariables over the predicate symbols. Each predicate symbol has an arity >0. Two special predicates are the unary predicate E (existence) and the binary predicate = (equality). There is a class of unary predicates, not including E, called *type predicates*. Type predicates are used to indicate basic kinds of things, like *Emp*, *Book*, *Dept* etc. (cf. Reiter [1984], p.195). Formulas are built in the usual way using $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\forall$, $\exists$, and punctuation symbols (,), [and]. We use infix notation for =. Metavariables over formulas are ϕ and ψ. The following abbreviations are used:

$$\forall^E x[\phi(x)] \overset{\Delta}{\Leftrightarrow} \forall x[E(x) \Rightarrow \phi(x)] \text{ and}$$
$$\exists^E x[\phi(x)] \overset{\Delta}{\Leftrightarrow} \exists x[E(x) \wedge \phi(x)].$$

Definition 2.

A function symbol f of arity $n>1$ is called *transparent* if for any constants $c_1, \ldots, c_n$, there is a constant c_0 such that $f(c_1, \ldots, c_n) = c_0$. □

If f is transparent then if the arguments of a particular application are known (in the sense of having a name), then the result of application is known. In any expression, function applications to constants can thus be eliminated.

Semantics

Although we have given the syntax of a first-order language without modal operators, we give a semantics in terms of a Kripke structure. In order to eliminate problems with unnamable objects in models, which may exist in the consistency view of IC satisfaction, we use universes in which all objects are named. Such universes can be built from the constants in the language by a Herbrand construction.

Definition 3.

For any language L,

1. the *Herbrand universe* U_L of L is the set of constants of L. (Since we consider only languages with transparent function symbols, it is sufficient to consider a Herbrand universe without function symbols.)
2. The *Herbrand base* $\mathcal{B}_L$ of L is the set of all ground atoms (closed atomic formulas) of L.
3. A *Herbrand model* $\mathcal{M}_L$ is a subset $\mathcal{M}_L \subseteq \mathcal{B}_L$. Truth in $\mathcal{M}_L$ is defined for ground atoms as

$$\mathcal{M}_L \models P(c_1, \ldots, c_n) \overset{\Delta}{\Leftrightarrow} P(c_1, \ldots, c_n) \in \mathcal{M}_L \text{ and } E(c_i) \in \mathcal{M}_L, i = 1, \ldots, n.$$

For an arbitrary closed ϕ, truth in $\mathcal{M}_L$ is defined in the usual way (e.g. see Lloyd [1984]). □

Intuitively, we may think of a Herbrand model as a set of KB-tuples, i.e. a single KB state. If we would want to describe the true facts in a Herbrand model by a theory, we would need a completion axiom for each predicate, stating that all and only the true facts in the Herbrand model are derivable (Reiter [1984]). In our model-theoretic view, the above truth definition plays an analogous role, for it says that all and only the tuples in the Herbrand model are true.

Definition 4.

An *S5 Herbrand-Kripke structure* $\mathcal{K}_L$ of a language L is a collection of Herbrand models which are called the *worlds* or *states* of $\mathcal{K}_L$. Truth of a ground atom in $\mathcal{K}_L$ is defined as

$$\mathcal{K}_L \models P(c_1, \ldots, c_n) \overset{\Delta}{\Leftrightarrow} w \models P(c_1, \ldots, c_n) \text{ for all } w \in \mathcal{K}_L.$$

Truth of a closed formula ϕ in $\mathcal{K}_L$ is then defined in the usual way. The collection of all Herbrand-

Kripke structures of L is called $\mathcal{L}$. □

We will drop the qualification "S5" from the definition from now on. A Herbrand-Kripke structure may be thought of as the collection of all possible KB states. This collection is the state space through which the KB moves during its existence. Each constant c denotes an object in a possible state of the KB and in each possible world w of the structure the existence predicate E denotes the set of existing objects in that world.

Static KB theories

Definition 5.

The *theory* T of $\mathcal{M} \in \mathcal{L}$ is the set of sentences which are true in $\mathcal{M}$. □

Note that a theory of $\mathcal{M}$ is just a set of IC's of $\mathcal{M}$. We do not emphatically include statements of ground atomic facts in our theory; in our possible worlds model, the axioms of T are necessary truths of the states of a model of T, whereas ground atomic facts vary per state.

We now define *KB theories* as theories which contain 1. the theorems of predicate logic, 2. axioms common to all KB domains, 3. axioms specific to a particular KB domain. The axioms common to all KB domains are an adaption of Reiter's [1984] well-known closure axioms to Kripke-structures and single out, for each L, a Herbrand-Kripke structure as unique model (up to isomorphy) of the theory.

Definition 6.

Given a language L and a model $\mathcal{M} \in \mathcal{L}$, a theory T of $\mathcal{M}$ is called a *static KB theory* if $T = FOL \cup HB \cup D$, where:

1. FOL is the set of all theorems of first order predicate logic,
2. HB is a set of closure and equality axioms defined below,
3. D is a set of sentences called a *domain theory*.

D is the set of IC's for a particular KB, while the other axioms are shared by all KB's. HB consists of the following sentences.

1. A *domain closure* axiom $\forall x[x=c_1 \vee x=c_2 \vee \ldots]$, where all and only the constants of L appear among the c_i.
2. *Unique name* axioms $\neg(c_1=c_2)$ for all constants c_i and c_j, $i \neq j$.
3. *Equality axioms*
   3.1. $\forall x[x=x]$.
   3.2. $\forall x, y[x=y \Rightarrow y=x]$.
   3.3. $\forall x, y, z[x=y \wedge y=z \Rightarrow x=z]$.
4. For each predicate P a *substitution axiom* $\forall \vec{x}, \vec{y}[P(\vec{x}) \wedge x_1=y_1 \wedge \ldots \wedge x_n=y_n \Rightarrow P(\vec{y})]$, where $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_n$.
5. For each unary predicate, excluding E and type predicates, the *existence axiom* for an n-ary predicate P is

$$\forall x P(x_1, \ldots, x_n) \Rightarrow E(x_1) \wedge \ldots \wedge E(x_n). \square$$

Comments:

1. The domain closure axiom forces us to choose our objects from the Herbrand universe of L. In the tuples of a KB we will thus find only constants from L. The unique name axiom eliminates

confusion between named objects and the equality axioms define = to be an equivalence relation which according to the substitution axiom is a congruence with respect to the predicate symbols. These axioms tell us that when we find several occurrences of the same constant in different tuples, we can expect these constants to denote the same object.

2. Reiter [1984] also has predicate completion axioms, which for each predicate state that a particular set of constants is precisely the extension of that predicate. Obviously, no such axiom is true in our Kripke structure, for in different states of the structure a predicate will have different extensions. Our definition of truth in a state of the structure plays the role of predicate completion in that state, for it says for each predicate that there is a set of tuples of constants (i.e. nameable objects) which is precisely the extension of that predicate.

3. The existence axioms are meant exclude models with states like $\{E(c_1), P(c_1, c_2)\}$, of which it is false to say that c_2 exists but it is true to say that $P(c_1, c_2)$.

4. Our inclusion of *FOL* in any KB theory is meant to imply that we use modus ponens as a proof rule

   $$\text{MP} \qquad \frac{\vdash \phi,\ \vdash \phi \Rightarrow \psi}{\vdash \psi}$$

5. To show that a KB theory is satisfiable, we construct a universe from the constants in the language and define extensions of E and other predicates in different worlds of the structure. Note that in general, if a theory has a Herbrand model then it has a model, but only for theories in clausal form (universally quantified conjunctions of disjunctions, possibly with Skolem functions) the implication works the other way as well. A theory in clausal form has a Herbrand model if it has a model at all. Since we do not bother to put our theories in clausal form, we explicitly construct Herbrand models.

We now distinguish the three different types tof heories described in this paper.

Definition 7.

1. If L is L_{Stat} and

   $$T = FOL \cup HB \cup D \text{ with}$$
   $$D = Stat,$$

   where *Stat* is a set of sentences in L_{Stat} called *static IC's*, then T is called a *static* KB theory and D a *static* domain theory.

2. If L is L_{Dyn} (the language defined in the next section) and

   $$T = FOL \cup HB \cup DL \cup D \text{ with}$$
   $$D = Stat \cup Dyn,$$

   where *DL* is the set of axioms of dynamic logic introduced in the next section and *Dyn* a non-empty set of sentences in L_{Dyn}, then T is called a *dynamic* KB theory and D a *dynamic* domain theory. *Dyn* is the set of *dynamic IC's* of D.

3. If L is L_{Dyn} and

   $$T = FOL \cup HB \cup DL \cup D \text{ with}$$
   $$D = Stat \cup Dyn \cup Deon,$$

   where *Deon* is a non-empty set of sentences in L_{Dyn} using violation predicates (defined in section 3), then T is called a *deontic* KB theory and D a *deontic* domain theory. *Deon* is the set of *deontic IC's* of D. □

4. Dynamic necessary constraints

We choose a variant of dynamic logic (DL, Harel [1984]) to express dynamic constraints because this will allow us to express deontic constraints as well. Temporal logic (TL), which has been used for dynamic constraint specification as well, has the drawback that it is not compositional. To give a semantics to a TL formula in terms of the process which the formula is talking about, one must "unroll" the process and quantify over the objects occurring in this unrolled process (e.g., Ehrich et al. [1984], Lipeck & Saake [1987]). If Φ_1 and Φ_2 are two temporal logic formulas interpreted in two processes, then if two processes are combined, e.g. in a parallel composition, the universe of objects into which the combined formula is interpreted differs from the two universes into which each formula is interpreted separately. This means that we cannot simply compose the semantics of the separate formulas but must unroll the combined process anew. Compositionality is a basic desideratum for any formalism to reason about processes. The axioms of DL and TL are both *syntax-directed* in the sense that they are given by breaking down the syntactic structure of a formula. But because the syntactic constructs of DL formulas describe the composition operations of processes (e.g. sequential, alternative, parallel composition), DL is compositional with respect to processes. The syntactic constructs of TL on the other hand do not correspond to composition operations on processes, which leads to the lack of compositionality of TL.

We start by defining a language for actions and then use this to extend the language L_{Stat} to a dynamic language L_{Dyn}.

Actions

We keep the language of actions as general as possible so as to accommodate diverse applications. We therefore assume that a countable set A of unspecified primitive actions, with metavariable a (possibly subscripted) ranging over A. We build composite actions out of primitive actions as follows.

Definition 8.

The language L_{Act} of actions, with typical elements α, is given by the following BNF:

$$\alpha \ ::= \ a \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \& \alpha_2 \mid \overline{\alpha} \mid \textbf{any} \mid \textbf{fail}$$

where $a \in A$. $\alpha_1 \cup \alpha_2$ is a non-deterministic choice of the actions α_1 and α_2; $\alpha_1 \& \alpha_2$ is the parallel execution/performance of the actions α_1 and α_2; $\overline{\alpha}$ is the non-performance of the action α; **any** denotes the unspecified action; **fail** denotes the failing (empty) action. □

Actions may change the world, and if they do, they do it instantaneously, i.e. there are no intermediate worlds during the execution of an action. The execution of an action is also called a *step*. If a UoD event spans a number of states, it must be represented by a transaction consisting of two or more steps (see below for transactions). In terms of our (Herbrand-)Kripke models, an action maps possible worlds to possible worlds, and an execution of an action in a world is the transition from that world to another world. The action **fail** has no successor worlds and the action **any** executed in a world has an arbitrary world as successor. For a formal semantics we refer to Meyer [1988]. Here it is sufficient to require that the domain of interpretation of L_{Act} is a Boolean algebra with respect to $\cup$, &, and $\overline{\ }$, with **fail** as zero and **any** as unit.

Actions are instantaneous in that they have a duration of one step. Transactions, on the other hand, can take more steps because they consist of sequences of actions.

Definition 9.

The language L_{Trans} of transactions, with typical elements β, is given by the BNF:

$\beta \ ::= \ \alpha \mid \beta_1;\beta_2 \mid \mathbf{clock}$

where $\alpha \in L_{Act}$. □

Intuitively, $\beta_1;\beta_2$ is the sequential composition of the transactions β_1 and β_2 and **clock** is a transaction of the duration of one time unit. We assume that a time unit has been chosen for the UoD, giving an intuitive interpretation to one tick of the clock. If the time unit is one day, then **clock** is the passing of one day, if the unit is one minute, then **clock** is the passing of one minute. During a tick of the clock, **any** is executed one or more times.

Definition 10.

The following abbreviations are used:

$$\beta^n \overset{\Delta}{=} \beta; \ldots; \beta \qquad \text{(n times)}$$

$$\alpha_{(n)} \overset{\Delta}{=} \mathbf{any}^n; \alpha \qquad \text{(note: } \alpha_{(0)} \equiv \alpha)$$

$$\alpha_{(\le d)} \overset{\Delta}{=} \alpha_{(0)} \cup \ldots \cup \alpha_{(d)}$$

$$\overline{\alpha}_{(n)} \overset{\Delta}{=} \mathbf{clock}^n; \overline{\alpha}$$

$$\alpha_{(>d)} \overset{\Delta}{=} \overline{\alpha}_{(\le d)} = \overline{\alpha}_{(0)} \& \ldots \& \ \overline{\alpha}_{(d)} \ \Box$$

Thus, in a library administration where *return* is a the action of returning a book and the time unit is one calendar week, $return_{(\le 3)}$ is the action of returning the book at the latest 3 weeks after now (= the moment that $return_{(\le 3)}$ is executed). $\overline{return}_{(3)}$ is the action of not returning the book in the third week from now, and $return_{(>3)}$ is any transaction not containing the action of returning the book within three weeks.

Dynamic constraints

We now extend the language for static constraints to a language for dynamic constraints. The language we define below is a variant of what is called PDL (Propositional Dynamic Logic) in the literature (Harel [1984]).

Definition 11.

The language L_{Dyn} of dynamic constraints, with typical elements Φ and Ψ, is given by the BNF:

$$\Phi ::= \phi \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \Rightarrow \Phi_2 \mid \Phi_1 \Leftrightarrow \Phi_2 \mid [\beta]\Phi \mid DONE{:}\alpha$$

where ϕ is a formula of L_{Stat}. □

A dynamic constraint (a formula in L_{Dyn}) is thus a static constraint, or a logical combination of dynamic constraints, or has the form $[\beta]\Phi$ or $DONE{:}\alpha$. $[\beta]\Phi$ is true in all those states where execution of transaction β necessarily leads to a state where dynamic constraint Φ holds. It is thus the weakest precondition of β with respect to the postcondition Φ. $DONE{:}\alpha$ expresses that the action α has been performed. It is true in all worlds where α has just been performed.

We use

$\langle\beta\rangle\Phi$

as an abbreviation of $\neg[\beta]\neg\Phi$, read intuitively as the statement that the execution of transaction β may lead to a state where Φ holds. Again, for a formal semantics we refer to Meyer [1988], [to appear]. Here, we limit ourselves to the following. Remember that $\mathcal{L}$ is the set of all S5 Herbrand-Kripke structures of L.

Definition 12.

Let $w \in \mathcal{M} \in \mathcal{L}_{Dyn}$. Then w satisfies $[\beta]\Phi$, written $w \models [\beta]\Phi$, iff

$w' \models \Phi$ for every $w' \in K_L$ such that the execution of β in w yields w'. □

In order to reason about dynamic constraints, we have to extend our domain theory.

Definition 13.

Given a model $\mathcal{M} \in \mathcal{L}_{Dyn}$, a theory T of $\mathcal{M}$ is called a *dynamic KB theory* if $T = FOL \cup HB \cup DL \cup D$, where FOL and HB are as in definition 6, D is a dynamic domain theory and DL is the set

DL1	$[\alpha](\Phi_1 \Rightarrow \Phi_2) \Rightarrow ([\alpha]\Phi_1 \Rightarrow [\alpha]\Phi_2)$
DL2	$[\alpha_1 \cup \alpha_2]\Phi \Leftrightarrow [\alpha_1]\Phi \wedge [\alpha_2]\Phi$
DL3	$[\alpha_1 \& \alpha_2]\Phi \Leftrightarrow [\alpha_1](DONE{:}\alpha_2 \Rightarrow \Phi)$
DL4	$[\,\overline{(\alpha_1 \cup \alpha_2)}]\Phi \Leftrightarrow [\bar{\alpha}_1 \& \bar{\alpha}_2]\Phi$
DL5	$[\,\overline{(\alpha_1 \& \alpha_2)}]\Phi \Leftrightarrow [\bar{\alpha}_1 \cup \bar{\alpha}_2]\Phi$
DL6	$[\mathbf{fail}]\Phi \Leftrightarrow \mathbf{true}$
DL7	$[\beta_1;\beta_2]\Phi \Leftrightarrow [\beta_1]([\beta_2]\Phi)$
DL8	$DONE{:}(\alpha_1 \cup \alpha_2) \Leftrightarrow DONE{:}\alpha_1 \vee DONE{:}\alpha_2$
DL9	$DONE{:}(\alpha_1 \& \alpha_2) \Leftrightarrow DONE{:}\alpha_1 \wedge DONE{:}\alpha_2$
DL10	$\overline{DONE}{:}\alpha \Leftrightarrow \neg DONE{:}\alpha$
DL11	$DONE{:}\mathbf{any} \Leftrightarrow \mathbf{true}$
DL12	$DONE{:}\mathbf{fail} \Leftrightarrow \mathbf{false}$
DL13	$[\alpha]DONE{:}\alpha$
DL14	$[\alpha_1]\Phi \Rightarrow [\alpha_2](DONE{:}\alpha_1 \Rightarrow \Phi)$
DL15	$t = n \Rightarrow [\mathbf{clock}]t = n+1$

Derivation rules are

MP	$\Phi, \Phi \Rightarrow \Psi \ \vdash\ \Psi$
N	$\Phi \ \vdash\ [\alpha]\Phi$

□

Remarks:

1. The axioms are schemata, to be instantiated for each metavariable α, β, ..., Φ,
2. DL2 says that a nondeterministic choice between α_1 and α_2 is guaranteed to lead to Φ iff both α_1 and α_2 necessarily lead to Φ.
3. DL3 holds because actions are instantaneous. If actions would have a duration of more than one step, then DL3 would hold only if α_1 and α_2 have the same duration.
4. DL6 is valid because Φ is vacuously true *after* the performance of **fail**, because **fail** has no successor states.
5. DL14 is valid because if α_1 necessarily leads to Φ, then if any action α_2 has been performed, Φ holds if α_1 has been performed as well.
6. In DL15 t is a distinguished variable which is increased by 1 every time the clock ticks. We intuitively interpret this as real time. Real time is included in order to express obligations in

deontic constraints (this is explained in the next section).

7. Rule N should be distinguished from the formula $\Phi \Rightarrow [\alpha]\Phi$, which is not valid in general. This formula expresses that Φ is an invariant under the execution of α, while N merely expresses that a valid formula Φ holds everywhere, so also after the performance of α.

8. For this system to be sound, we must require that atomic actions are unique in the sense that every atomic action leaves a unique marking in the world immediately resulting from its execution, such that it can be uniquely determined whether the action has just been performed. We do this by means of the predicate $DONE{:}\alpha$, which is true in worlds which result from the execution of α and false in other worlds. (So in practice, several occurrences of the "same" atomic action must be labeled in order to distinguish them.)

We left open the structure of the set A of primitive actions. Usually primitive actions will be parameterized, so that executions with different actual parameters will have different effects on the state of the world. For example, let $Salary(e, n)$ express that employee e has salary n, then

S0 $\quad \forall^E e, s, n[Salary(e, s) \Rightarrow [change\text{-}salary(e, n)]Salary(e, s{+}n)]$.

says that for each value of p and n, $change\text{-}salary(p, n)$ is a primitive action which has the effect of adding n to the salary of p. Note that the effect of the action is only defined for existing objects. We can add typing information to the axiom as a precondition,

S1 $\quad \forall^E e, s, n[Emp(e) \wedge Salary(n) \wedge Salary(e, s) \Rightarrow [change\text{-}salary(e, n)]Salary(e, s{+}n)]$.

Alternatively, we can add the axiom

$$\forall^E e, s, n[Salary(e, s) \Rightarrow Emp(e) \wedge (Salary(s)]$$

and omit typing information from the dynamic axiom.

5. Deontic constraints

We need no extensions to L_{Dyn} to be able to express deontic constraints. The deontic concepts of obligation and permission can be reduced to the concept of prohibition, which in turn can be reduced to the concept of an action leading to a *violation* of a rule. Instead of expressing the rules explicitly, we thus state when they are violated. We do this by defining, for each action α, one or more *violation states* $V_i{:}\,\alpha$, one for each of the reasons why the execution of α is forbidden. For each violation state, we usually define a *corrective action* which allows one to get out of that state. The necessary reductions are then effected by the following definition.

Definition 14.

The following abbreviations are used.

$F(\alpha) \overset{\Delta}{\Leftrightarrow} [\alpha]V_i{:}\alpha$ for an i,

$P(\alpha) \overset{\Delta}{\Leftrightarrow} \neg F(\alpha)$ and

$O(\alpha) \overset{\Delta}{\Leftrightarrow} F(\overline{\alpha})$.

$F(\alpha)$ is pronounced "α is forbidden", $P(\alpha)$ is pronounced "α is permitted", and $O(\alpha)$ is pronounced "α is obligatory." □

We thus consider an event forbidden if it necessarily leads to a violation state of that action. The reduction of prohibitions to actions has been first proposed by Anderson [1967] and has been first formalized in the context of dynamic logic by Meyer [1988]. The use of dynamic logic enables the separation of actions from states, which allows one to solve numerous paradoxes of deontic logic

(Meyer [1987], [1988]).

An action is permitted iff it is not forbidden, which is equivalent to saying $P(\alpha) \Leftrightarrow <\alpha>\neg V{:}\alpha$. An action is permitted if there is at least one instance of doing it which leads not to a state of violation. Finally, an action is obligatory iff not doing it is prohibited, i.e. iff $[\overline{\alpha}]V{:}\overline{\alpha}$.

Note that there is a practical difference between a prohibition and an obligation. The violation of a prohibition can be observed immediately: if one is forbidden to steal a book from a library, the violation of this prohibition can be established as soon as theft is committed. On the other hand, when one is obliged to return a book borrowed from a library, the violation of such an obligation cannot be determined when no term is set in which the performance of the obliged action, cq. the return of the book, has to occur. Therefore, in our examples, we shall only use obligations which must be fulfilled within a specific interval of time after the obligation is incurred.

Definition 15.

Given a model $\mathcal{M} \in \mathcal{L}_{Dyn}$ and a theory $T = FOL \cup HB \cup DL \cup D$, where FOL, HB, DL and D are as in definition 13, then T is called a *deontic KB theory* $D = Stat \cup Dyn \cup Deon$, with *Stat* and *Dyn* (possibly empty) defined as in definitions 6 and 13 and *Deon* a non-empty set of dynamic formulas containing violation predicates $V_i{:}\alpha$. □

An example of a deontic axiom is

S2 $\quad \forall^E e, s, n[Salary(e,s) \wedge n>s \Rightarrow [change\text{-}salary(e,n)]V_1{:}salary\text{-}change(e,n)]$.

This axioms says that it is forbidden to double a salary in one action. If such an action is attempted, a violation state is entered. Note that the action parameters are also parameters of the violation state, so that sufficient information is available for a corrective action. Assuming that S1 is also present, execution of $salary\text{-}change(e_1, 1000)$ in state $Salary(e_1, 300)$ leads to a state $Salary(e_1, 1300) \wedge V_1{:}salary\text{-}change(e_1, 1000)$. A possible corrective action to this state could be

S3 $\quad Salary(e,s) \wedge V_1{:}salary\text{-}change(e,n) \Rightarrow [salary\text{-}change(s,-n)]\neg V_1{:}salary\text{-}change(e,n)$.

S1 guarantees that after this corrective action the salary has been changed appropriately.

Using violation states, one has the choice of modeling rules for the UoD as necessary truths or as deontic constraints. For example, if a bank account may not be negative, we can represent this in the domain theory in one of the following two ways:

A1.1 $\quad Balance(a,n) \wedge n+m<0 \Rightarrow [update\text{-}balance(a,m)]Balance(a,n)$

A1.2 $\quad Balance(a,n) \wedge n+m\geq 0 \Rightarrow [update\text{-}balance(a,m)]Balance(a,n+m)$

or

A2.1 $\quad Balance(a,n) \Rightarrow [update\text{-}balance(a,m)]Balance(a,n+m)$

A2.2 $\quad Balance(a,n) \wedge \neg V{:}update\text{-}balance(a,m_1) \wedge n+m_2<0 \Rightarrow [update\text{-}balance(a,m_2)]$
$V{:}update\text{-}balance(a,m_2)$

A2.3 $\quad Balance(a,n) \wedge V{:}update\text{-}balance(a,m_1) \wedge n+m_2<0 \Rightarrow [update\text{-}balance(a,m_2)]$
$V{:}update\text{-}balance(a,m_1+m_2)$

A2.4 $\quad Balance(a,n) \wedge V{:}update\text{-}balance(a,m_1) \wedge n+m_2\geq 0 \Rightarrow [update\text{-}balance(a,m_2)]$
$\neg V{:}update\text{-}balance(a,m_1)$

A1 never allows a balance to drop below zero, A2 allows it to drop below zero (A2.1) but signals that this is a violation state when it occurs (A2.2), remembers the extent of the violation (A2.3) and provides a way of correcting it (A2.4).

In practice, banks combine A1 and A2 by allowing an account to be negative but not less than a certain amount. In that case A2.2 and A2.3 are modified by adding the test $Permissible\text{-}overdraw(a, o) \wedge o<n+m_1$ as a precondition and adding A3:

A3 $Balance(a, n) \wedge Permissible\text{-}overdraw(a, o) \wedge n+m<o \Rightarrow [update\text{-}balance(a, m)]$ $Balance(a, n) \wedge O(refuse(a, m))$.

$refuse(a, m)$ is the explicit action of refusing an account update.

We end by making some philosophic observations about the system *Deon*. First, note that there are three important reductions in this system, which need not be made at the same time. The first reduction is that of deontic logic to dynamic logic. Given this reduction, we can distinguish between actions and states and make the second reduction to reduce prohibitions, which are properties of actions, to violations, which are properties of states. It is this second choice which makes our system a *reductionistic value system* (cf. Huisjes [1981]). Another choice would have been possible, in which an action is not forbidden because it leads to punishement, but because it is intrinsically bad. For example, it may be forbidden because the scripture says it is one of a set of prohibited actions, or because it contradicts the golden rule "do as thou would be done to," or because it is not performed in the proper way. In all these cases, prohibition is a property of the action itself and not of the state resulting from the action.

Independently of the first two choices, we can thirdly choose to reduce nonpermissions to prohibitions. This choice results in a *closed* value system, by which is meant that every action is deontically determined: for each α,

$$\vdash P\alpha \vee F\alpha.$$

This is not a default assumption about *which* of the two is true, $P\alpha$ or $F\alpha$. Addition of such an assumption would lead to nonmonotonic phenomena (e.g. Etherington [1988]).

6. Conclusion

In the introduction we distinguished necessary from deontic constraints and noted that empirical constraints, if formulated weak enough, can for the purpose of KB design be treated as necessary constraints. Violation of necessary constraints is impossible in the UoD and, when they occur in the implementation, are implementation errors. Deontic constraints must be treated differently from necessary constraints. Violation of a deontic constraint does occur in the UoD and must be represented by the KB. In sections 2-5, a particular formalism for describing necessary as well as deontic truths about the UoD was presented. In general, the deontic variant of dynamic logic has the merits that it avoids certain paradoxes in deontic logic. For the specification of KB's it has the additional advantage of allowing to specify static, dynamic and deontic constraints in a single coherent framework.

The implementation of integrity constraints is not covered in this paper. An attempt to implement a conceptual language that includes deontic operators is reported in (Dignum et al. [1987]). In this implementation, a distinction is made between deontic constraints whose satisfaction can be enforced by the system and those which cannot be so enforced. For example, a bank account system can enforce the constraint that a client is not allowed to withdraw any money from an account when the balance has fallen below a certain negative amount, but it cannot enforce the obligation that the customer must pay in sufficient money for the balance to be positive again.

The distinction between necessary and deontic constraints, and within deontic constraints between enforcable and non-enforcable constraints, is also made in the ISO report on conceptual schema terminology (Griethuysen [1982], section 2.5) However, the ISO report does not use the results of contemporary analytic philosophy to explicate these concepts clearly and offers no logic to express the different types of constraint, as we do (cf. Hospers [1953], Moser [1987], Munitz [1981]).

Deontic logic is used by Lee [1988] to specify obligations, prohibitions and permissions in an office environment. Lee also stresses the performative aspects of office information systems. However, he employs a deontic logic based on Anderson's reduction to alethic modal logic (Anderson [1967], Hilpinen [1988a, b]), which has been shown by McArthur [1981] to contain a number of paradoxes. The deontic variant of dynamic logic which we use does not suffer from these paradoxes (Meyer [1987], [1988]) and has the added advantage that it can be embedded smoothly in our language for dynamic constraints.

One topic left open in our research is the inheritance of constraints in taxonomic hierarchies. Are all prohibitions, permissions and obligations of members of a superclass also prohibitions, permissions and obligations of members of a subclass? Does a manager have more or less obligations than an employee?

A second cluster of open problems circles around constraint satisfiability. The satisfiability problem for IC's is the question

1. Is there a non-empty set of closed Herbrand models such that all axioms of the domain theory are satisfied in that set?

We have not shown how this question can be answered in general. Other, more interesting questions for KB modeling are

2. Is there a model such that in each world there is at least one executable action, i.e. an action leading to a world? If not, there are "black holes," worlds from which there is no escape.
3. For each action, is there a world in which it can be executed? If not, the action is redundant.
4. For each predicate, is there a world in which it has a non-empty extension? If not, the predicate is redundant.
5. For any world in which at least one violation predicate has a non-empty extension, is there an action applicable which will diminish this extension? If not, some violations, once committed, cannot be undone.

We plan to tackle these questions in the future.

References

Anderson A.R. [1967]

"Some Nasty Problems in the Formalization of Ethics," *Noûs*, Vol. 1 (1967), 345-360.

Dignum F., T. Kemme, W. Kreuzen, H. Weigand, R.P. van de Riet [1987]

"Constraint Modelling Using an Conceptual Prototyping Language," *Data & Knowledge Engineering*, Vol. 2, 213-254.

Ehrich H.-D., U.W. Lipeck, M. Gogolla [1984]

"Specification, Semantics, and Enforcement of Dynamic Database Constraints," *Proc. of the Tenth International Conference on Very Large Databases*, Singapore, August 1984, 301-308.

Etherington D.W. [1988]

Reasoning with Incomplete Information, Pitman.

Fiadeiro J, A. Sernadas [1988]

"Specification and Verification of Database Dynamics," *Acta Informatica* 25, 625-661.

Griethuysen J.J. van (ed.) [1982]

Concepts and Terminology for the Conceptual Schema and the Information Base, ISO TC97/SC5/WG3 Report.

Harel D. [1984]

"Dynamic Logic," in: D.M. Gabbay, F. Guenther (eds.), *Handbook of Philosophical Logic, Vol. 2,* Reidel.

Hilpinen R. (ed.) [1988a]

Deontic Logic: Introductory and Systematic Readings, Reidel.

Hilpinen R. (ed.) [1988b]

New Studies in Deontic Logic, Reidel.

Hospers J. [1953]

An Introduction to Philosophical Analysis, Prentice-Hall.

Hughes G.E., M.J. Cresswell [1968]

An Introduction to Modal Logic, Methuen.

Huisjes C.H. [1981]

Norms and Logic, Ph.D. Thesis, Rijksuniversiteit te Groningen.

Kung C. [1985]

"A Tableaux Approach for Consistency Checking," in: A. Sernadas, J. Bubenko, A. Olivé (eds.), *Information Systems: Theoretical and Formal Aspects,* North-Holland, 191-207.

Lee R.M. [1988]

"Bureaucracies as Deontic Systems," *Trans. on Office Information Systems,* Vol. 6, no. 2, 87-108.

Lipeck U.W. [1986]

"Stepwise Specification of Dynamic Database Behaviour," *Proc. SIGMOD,* 387-397.

Lipeck U.W., G. Saake [1987]

"Monitoring Dynamic Integrity Constraints Based on Temporal Logic," *Information Systems,* Vol. 12, no. 3, 225-269.

Lloyd J.W. [1984]

Foundations of Logic Programming, Springer.

Lloyd J.W., E.A. Sonenberg, R.W. Topor [1987]

"Integrity Constraint Checking in Stratified Databases," *Journal of Logic Programming,* 4, 331-343.

McArthur R.P. [1981]

"Anderson's Deontic Logic and Relevant Implication," *Notre Dame Journal of Symbolic Logic,* Vol. 22, 145-154.

Meyer J.-J Ch. [1987]

"A Simple Solution to the ''Deepest'' Paradox in Deontic Logic," *Logique et Analyse,* Vol. 30, 81-90.

Meyer J.-J. Ch. [1988]

"A Different Approach to Deontic Logic: Deontic Logic Viewed As a Variant of Dynamic

Logic," *Notre Dame Journal of Formal Logic 19(1), 109-136.*

Meyer J.-J. Ch. [to appear]

"Using Programming Concepts in Deontic Reasoning," to appear in: R. Bartsch, J. van Benthem, P. van Emde Boas (eds.), *Semantics and Contextual Expression,* FORIS Publications, Dordrecht-Riverton.

Moser P.K. (ed.) [1987]

A Priori Knowledge, Oxford University Press.

Munitz M.K. [1981]

Contemporary Analytic Philosophy, MacMillan.

Nicolas J.M. [1982]

"Logic for Improving Integrity Checking in Relational databases," *Acta Informatica* 18, 227-253.

Nicolas J.M., H. Gallaire [1978]

"Data Base: Theory vs. Interpretation," in Gallaire & Minker [1978], 33-54.

Nicolas J.M., K. Yazdanian [1978]

"Integrity Checking in Deductive Databases," in Gallaire & Minker [1978], 325-344.

Reiter R. [1984]

"Towards a Logical Reconstruction of Relational Database Theory," in: M. Brodie, J. Mylopoulos, J. Schmidt (eds.), *On Conceptual Modelling,* Springer,191-233.

Reiter R. [1988]

"On Integrity Constraints," in M.Y. Vardi (ed.), *Proc. of the Second Conf. on Theoretical Aspects of Reasoning about Knowledge,* Morgan Kaufmann, 97-111.

Sernadas A. [1980]

"Temporal Aspects of Logical Procedure Definition," *Information Systems,* 5, 167-187.

Weber W., W. Stucky, J. Karszt [1983]

"Integrity Checking in Data Base Systems," *Information Systems,* 8, 125-136.

Wieringa R.J., R.P. van de Riet [1988]

"Algebraic Specification of Object Dynamics in Knowledge Base Domains," in *Proc. of the IFIP TC2/WG 2.6 and TC8/WG8.1 Working Conf. on the Role of Artificial Intelligence in Databases and Information Systems,* Canton, China, 4-8 july, 1988.

Wright, G.H. von [1963]

Norms and Action, Routledge and Kegan Paul.

Appendix: A Library KB

The UoD of this example is a library which contains 2000 works and has 750 members. A member can borrow or return one or several works by applying to one of the library wickets. S/he also has the possibility to reserve a work if none of its copies are available. In that case, his or her reservation is placed at the end of a queue of reservations made for the same work. As soon as a copy is returned, the first member in the queue is informed that the work is available. The book is then kept during one week for this member, after which it is free again to be borrowed by the next member in the queue or, if the queue is empty, by any member of the library.

A library member cannot have more than 3 books at a time and each loan has to be returned at the end of 3 weeks. If the book is not returned, the library will send a reminder. As long as the book is not returned, the member cannot borrow other works. The charge for returning a book too late is \$2.

Signature

Constants: $\{Self, \$2, B_1, B_2, \ldots, B_{2000}, P_1, P_2, \ldots, P_{750}, 0, 1, 2, 3, \ldots\}$

Type predicates (for each type predicate a typical variable is given):

$Natural(n)$, n is a natural number
$Person(p)$, p is a person
$Library(l)$, l is a library
$Book(b)$, b is a book
$Money(m)$, m is an amount of money

Other predicates:

$Available(b)$, b is not borrowed and not reserved
$Present(b)$, b is not borrowed
$Member(p)$, p is a member of the library
$PERF{:}\alpha$ for each of the actions below except *reserve*: A has been performed in the past
$PERF{:}reserve(p, b, n)$, p is nth in the list of reservers of b
$V{:}\alpha$ for each of the actions below below)

Functions:

$max(x, y)$, a function which gives the maximum of two numbers. (Note that this is a transparent function.)

Actions: (We use the convention that the agent of an action, if there is any, is the first argument of the action and is separated from the other arguments by a semicolon).

$borrow(p;b)$, p borrows b
$return(p;b)$, p returns b
$reserve(p;b)$, p reserves b
$notify(l;p,b)$, the library notifies p that b is available
$pay(p;m,b)$, p pays m concerning a book b

If $X = \{x \mid P(x)\}$ and $x \in X$ is another way of writing $P(x)$, we use the following abbreviation for the cardinality of X in a world,

$$card(X)=n \overset{\Delta}{\Leftrightarrow} \exists_n^E x{\in}X.$$

$\exists_n^E x{\in}X$ means that there are precisely n different elements in X,

$$\exists_n^E x\ Px \Leftrightarrow$$
$$\exists^E y_1,\ldots,y_n{:}\ y_1{\neq}\ldots(pairwise)\ldots{\neq}y_n \wedge \forall^E x\ [Px \Leftrightarrow x{=}y_1 \vee \ldots \vee x{=}y_n].$$

Assuming that we start with an empty extension for E, all reachable worlds will have a finite extension for E, so that the existential quantifier in the last formula can be written as a finite disjunction.

Necessary static constraints

IC0 $Library(Self), Money(\$2), \ldots, Book(B_1), \ldots, Natural(0), \ldots$

IC1 $\forall b \; [Available(b) \Leftrightarrow \forall p, n \; (Present(b) \wedge \neg PERF:reserve(p,b,n)]$

IC2 $\forall p_1,b \; [first\_reserver(p_1,b) \Leftrightarrow$
$\exists n \; [PERF:reserve(p_1,b,n) \wedge \forall p_2,n' : PERF:reserve(p_2,b,n') \Rightarrow n' = max(n', n)]$

IC0 introduces the constants. It also specifies that the UoD is described from the perspective of the library. The constant *Self* has no special logical meaning, but gets a special operational meaning when the specification is used as a prescription for the action component of the Library Information System.

Necessary dynamic constraints

IC3 $DONE:\alpha \Rightarrow [\beta]PERF:\alpha$

IC4 $PERF:\alpha \Rightarrow [\beta]PERF:\alpha$ for $\alpha \neq reserve$

These two axioms say that if an action has been done, it has been performed, and that once it has been performed, it remains in the state of having been performed. Contrast this with $DONE:\alpha$, which is only true in worlds resulting from α and false in other worlds.

IC5 $\forall p,b \; \neg PERF:borrow(p,b) \Rightarrow [return(p;b)] \, false$

IC6 $\forall p,b \; \neg Present(b) \Rightarrow [borrow(p;b)] false$

IC7 $\forall p,b,n \; [borrow(p;b)] \; \neg PERF:reserve(p,b,n) \wedge \neg Present(b)$

IC8 $\forall p,b \; [return(p;b)] \; Present(b) \wedge \neg PERF:borrow(p,b)$

IC9 $\forall p,b,n \; card(\{p_2 | \exists n' \; PERF:reserve(p_2,b,n')\}) = n$
$[reserve(p;b)] \; PERF:reserve(p,b,n+1)$

IC10 $\forall p,b,n \; [notify(Self;p,b)] \; [\mathbf{clock}^{(7)}] \; \neg PERF:reserve(p,b,n)$

Remarks:

1 IC5 and IC6 describe necessary preconditions. The other constraints all deal with the effects (postconditions) of actions.

2 According to IC10, a reservation is automatically cancelled when someone has failed to come and borrow the reserved book. This is an application of the *performative hypothesis*. When the reservation is cancelled in the data base, it is cancelled in reality. Therefore we do not need an extra action "cancel_reservation".

3 The implicit assumption of IC10 is that the communication between library and members is perfect so that the act of sending a notification is equivalent to the act of notifying.

Deontic constraints

IC11 For each action α, $O(\alpha_{(\leq n)}) \Rightarrow \alpha_{(m)} \wedge m > n \Rightarrow V:\alpha$.

IC12 $\forall p,b_1 \; P(borrow(p;b_1)) \Leftrightarrow [Member(p) \wedge \neg \exists b_2 \; V:\overline{return(p,b_2)}$
$\wedge \; card(\{b_3 \mid PERF:borrow(p,b_3)\}) < 3$
$\wedge \; ((Available(b_1) \vee first\_reserver(p,b_1))]$

IC13 $\forall p_1,b,n \; P(reserve(p_1;b)) \Leftrightarrow \; Member(p_1) \wedge \neg Available(b)$

IC14 $\forall p,b \; [borrow(p;b)] \; O(return(p;b)_{(\leq 21d)})$

IC15 $\forall p,b \; [borrow(p;b)] \; [\mathbf{clock}^{(21)}]$
$\neg PERF:return(p,b) \; \Rightarrow O(remind(Self;p,b))$

IC16 $\forall p,b \;\; first\_reserver(p,b) \wedge Present(b) \Rightarrow O(notify(Self;p,b))$

IC17 $\forall p,b \;\; V{:}return(p,b) \Rightarrow O(pay(p;\$2,b))$

IC18 $\forall p,b \;\; [borrow(p;b)]\ [\mathbf{clock}^{(21)}]\ [\neg PERF{:}return(p,b) \Rightarrow V{:}\overline{return(p,b)}]$

IC19 $\forall p,b \;\; [return(p;b)]\ \neg V{:}\overline{return(p,b)}$

IC20 $\forall p,b \;\; [pay(p;\$2,b)]\ \neg V{:}return(p,b)$

Remarks:

1. IC12 and IC13 define the permissions of the members of the library. IC12 says when it is permitted that someone borrows a book and IC13 says when it is permitted that someone reserves a book. Another reasonable permission would be that someone may only pay the library when he *must* pay: O(pay) $\Leftrightarrow$ P(pay). However, this constraint was not in the original description.
2. IC14-15 state several obligations. IC14 says that someone is obliged to return the book in three weeks (we assume the time unit of this UoD is one calendar day; see comment on definition 14). IC15 and IC16 state some obligations of the library: that the library should notify a reserver when the book becomes available and that it should send a reminder when a book is not returned in time. IC17 shows how the failure to perform an obliged action can lead to another obligation: if someone has not returned the book in time, he must pay a fine.
3. IC18-IC20 describe postconditions of actions as far as liability is concerned. IC18 specifies that someone is liable if he has not returned a book in time. This liability has some consequences (IC12: he can not borrow a new book). Note that this liability is cancelled (IC19) as soon as he returns the book (be it too late). However, when he returns the book too late, he performs a forbidden action (IC14) which leads to a liability $V{:}return(p,b)$. This liability is cancelled, according to IC20, when the offender pays a fine.
4. Note that the specification is not in all senses complete. For example, it is not said what happens when someone fails to pay the fine (IC17).