Data & Knowledge Engineering 4 (1989) 157–189
North-Holland

Specifying dynamic and deontic integrity constraints

Roel WIERINGA, John-Jules MEYER and Hans WEIGAND

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

Abstract. In the dominant 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 natural number (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 constrain the UoD. Instead, it constrains the implementation of the KB. We argue that the distinction between necessary and deontic IC's is relevant for KB modeling and that it imposes a more complicated modeling discipline on the KB designer than hitherto realized. 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 simple example to illustrate the difference between dynamic and static specification of deontic IC's, and a non-trivial example of a KB specification with static, dynamic and deontic constraints.

Keywords. Constraints, Deontic logic, Dynamic logic, Conceptual modelling.

0. Introduction

In current research in data- and knowledge bases, a number of different model concepts are used. The oldest and most explicit is the concept of a model as a *database schema*, which is roughly a set of relation definitions with some integrity constraints. We argue that this view is insufficient for an adequate representation of IC's, especially when the knowledge base is used to represent rules which the UoD should obey. By adopting the standard view of models from the philosophy of natural science, we show that some IC's currently encountered in the literature are analogous to analytical truths or to empirical laws of nature, but that many IC's cannot be so classified. Where analytical and empirical truths can be used to constrain the implementation in the sense that the implementation cannot be in a state (or cannot execute a state transition) which violates these constraints, there is a large class of other IC's which constrain the UoD, not the implementation of the KB. An example of such a constraint is that a library user ought to return a book or renew the lending period within three weeks of borrowing the book. This is a constraint on the UoD which can be violated by the UoD and the implementation of the KB must be able to represent such a violation. We call this class of constraints *deontic* ($\delta\varepsilon o\nu\tau\omega\varsigma$ (Greek) = "as it should be, duly").

In section 1.1 we argue that the standard descriptive model concept as it is used in natural science is applicable to KB's. We then show in Section 1.2 that this allows us to distinguish six types of IC in terms of the states and events obtaining in the UoD and which therefore are completely implementation-independent. Four types of IC can be defined wholly in terms of the standard models concept of natural science. Two other types require an

extension of this model concept with prescriptive elements, as shown in Section 1.3. In 1.4 we briefly point out some simplifications we make with respect to social UoD's

While Chapter 1 is language-independent, Chapter 2 shows how to express the six types of constraint in a deontic extension of dynamic logic. Section 2.1 formally defines our view of constraint satisfaction, and Sections 2.2–2.4 define a language for the different types of constraints distinguished in Chapter 1. In Chapter 3 we summarize the results and point out some topics for future research.

1. Descriptive and prescriptive models in knowledge base systems

1.1. Descriptive models

To understand the classification of integrity constraints in static constraints, dynamic constraints and deontic constraints we must have clear view of the role of models in the design and use of knowledge bases (KB's). The view of models presented in this section is mostly taken from natural science. To distinguish it from another type of model to be discussed below, we will call this type of model *descriptive*. Although the nature and role of descriptive models in natural science is not fully understood,[1] it suffices for our purposes to adopt the following definition.

Definition 1. A *descriptive model* of a UoD is an abstract system which represents the entities in the UoD and their behavior at a certain level of approximation. The system has a *state space* and a *state transition function* which describes how it can move through its state space.

Several concepts in this definition need elucidation.

1. The UoD of a knowledge base (KB) is usually a social system like (part of) an enterprise but may also be a technical reality like a system of elevators.
2. A descriptive model *represents* the UoD. What it is to represent part of reality is left unexplicated in this paper. It essentially involves being able to say that the model is *true* of the UoD or that the representation is *correct*, given a certain level of abstraction (cf. the references in note 1).
3. A descriptive model is *abstract*, by which is meant that parts of the UoD are not modeled at all, and that of those parts which are modeled irrelevant details are not represented. Put differently, the model approximates the UoD to a certain degree, bu looked at close enough discrepancies will be found where the model can be said to be an incorrect or incomplete representation of the UoD. In natural science, one studies ideal point masses and ideal gases which represent real masses and gases only to a degree of approximation. In KB's, a similar idealization takes place. For example, if we must represent the color of John's eyes as either blue or green, but they actually are blue-green, then we must approximate the real-world situation by representing John as having either blue or green eyes. Neither representation is correct, if we take the difference between blue, blue-green and green into account. And if we do not represent all of John's properties (KB's will in general not represent all properties of real-world objects) then in this sense the KB is an incomplete model of the UoD.
4. We view a model as a *system* in the sense of systems theory, which in the context of KB applications is roughly equivalent to the concept of machine in automata theory. That is, a model is a set of *state variables* whose values define a point in a state space and a

[1] See Harré [17], Hesse [18], Nagel [39], Suppe [49] and Suppes [50] for a discussion.

state transition function which assigns a (*next-state*, *output*) pair to each (*state*, *input*) pair. (In nondeterministic systems there is a set of (*next-state*, *output*) pairs for each (*state*, *input*) pair.). We only look at discrete-valued state variables, so that we may talk of a state *transition* function (as opposed to a state evolution function).

In the database community what is meant by "model" is usually a database schema, e.g. a set of relation definitions and integrity constraints. A relation schema comes close to what is called a *signature* in formal logic, a list of symbols used in a formal language L and for each predicate symbol its arity and possibly the types of its arguments. By contrast, we use the concept of a model primarily as it is used in natural science.[2] In this sense, each UoD entity e is represented by an abstract object o which is a model of e. (Speaking of the UoD, we talk about *entities* and speaking of abstract representations of the UoD, we talk about *objects*). In Fig. 1, the abstract objects o_1 and o_2 represent employees e_1 and e_2, so they are models of e_1 and e_2. Usually, the state variables of an object are called its *attributes*. The attributes of an object span a state space of dimensionality n, where n is the total number of attributes applicable to the object. At a higher level of aggregation, the KB itself is a model of the UoD because it is a set of abstract objects, each of which is a model of a UoD entity. Thus $\{o_1, o_2\}$ is a database which represents information about e_1 and e_2. A state of the database is characterized by giving 1. the set of abstract objects which exist in that database state and 2. the state of each existing object. The assumption here is that the UoD is in states which can be abstractly represented by states of the model and similarly that the entities in the UoD have states which can be represented by states of abstract objects.

Each system has a state transition function, and representation of object dynamics is currently an active research topic. To specify the state transition function of a model, we must specify update events (single state transitions) and processes (composed of sequences, choices, or parallel executions of update events). When a set of objects forms a database, they usually exchange messages, which greatly complicates the specification of the state transition function of the database. Wieringa and van de Riet [5] show how to use process algebra to specify the processes executed by individual objects as well as by a database (see Bergstra and Klop [3, 4] for process algebra). Below we will use dynamic logic to specify object and database dynamics.

We distinguish in Fig. 1 between the *model* of a UoD, which is an abstract KB and an

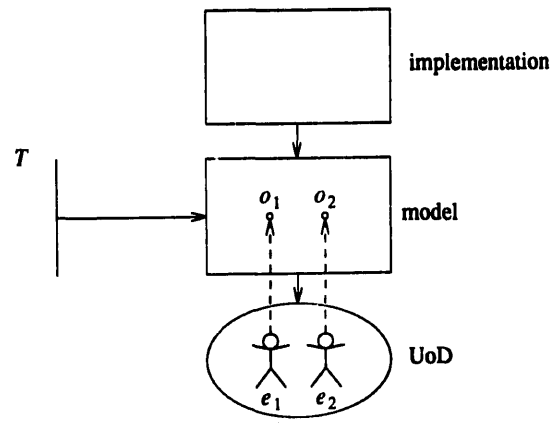

Fig. 1. Objects and entities.

[2]Other uses of the word "model" are: 1. a class of signatures with a certain structure, as for example in "the relational data model," "the functional data model;" 2. a particular state of a model in our sense of the word, i.e. a instance of a database schema (snapshot of a database).

implementation of the model, which is a concrete KB on a particular machine. The arrows in Fig. 1 will be explained later. In an implementation, an object is represented by a *record* consisting of a unique *key* and a *tuple* of attribute values. In this paper we concern ourselves with KB's as models of the UoD.

The physical sense of "model" can be connected with the logical sense of "model" which is common in database theory (Nicolas and Gallaire [41]). The model in Fig. 1 is an abstract structure into which a formal language L is interpreted. There is a set T of sentences in L which are true of the model, which is called a *theory* of the model. Following Reiter [44], we will use theories which have as only model their term-model divided by a suitable equivalence relation. That model is an abstract representation of the UoD. We thus take a model-theoretic view of KB's (as opposed to a KB as a set of sentences true to a model), taking the word "model" in two senses: as abstract representation of the UoD and as logical model of a set of sentences. Below we will extend this with a third sense, a model as a *prescription* for a UoD.

Our language L will be a deontic extension of dynamic logic (Harel [16], Meyer [34, 35, 36]), and our models will all be Kripke structures. Using the jargon of modal logic, the state space of a KB will often be called a set of *possible worlds* and a particular KB state is one possible world (Hughes and Cresswell [22]). We quantify over a single domain of possible objects, and we use the *existence predicate* E to express existence of an object in a world. If one uses an existence predicate E, to give the state of a KB one should give 1. the extension of E and 2. the extension of all other predicates and/or functions. A state transition function can be defined as an accessibility relation between worlds.

1.2. Integrity constraints

Definition 2. For a given logical language L and a class $\mathscr{L}$ of intended models, an *integrity constraint* (*IC*) of a KB $\mathcal{M} \in \mathscr{L}$ is a sentence ϕ in L such that $\mathcal{M} \models \phi$.

We again explain the main points in this definition.

1. The class of intended models of L are all Kripke structures consisting of a set of possible worlds (states). These states are *not* abstractions of machine states, but of UoD states.
2. A *sentence* is a closed formula. We thus take a very liberal view of IC's, since even an atomic sentence (of the form $P(c_1, \ldots, c_n)$ where P is a predicate and c_i are constants of L) is an IC if it is true in $\mathcal{M}$. But note that $\mathcal{M} \models P(c_1, \ldots, c_n)$ iff it is true in all states of the model. Our liberal view of IC's includes also constraints like $P(x_1, \ldots, x_n) \vee \neg P(x_1, \ldots, x_n)$, but at least it does not exclude anything from the status of IC which we should include.
3. ϕ is called a *constraint* only for historical reasons. As will be shown in a moment, some IC's constrain the implementation of a KB but not the KB itself, others constrain the KB but not the implementation, and still others do not constrain anything but are empirical generalizations.

Our definition is the model-theoretic analogue of the standard view of IC's (Kung [27], Nicolas and Gallaire [41], Nicolas and Yazdanian [42], Nicolas [40], Reiter [44]). What is not commonly realized in the literature is that this view implies that an IC is a *necessary truth* in the model. In modal logic a necessary truth is defined as truth in all possible worlds. This seems to be at odds with the status of a constraint like $age < 150$, which is not necessarily true in the UoD at all. Again, the statement that each course is taught by one teacher only is not a necessary truth either, since it can be violated, so by the above definition it is not an integrity constraint. These two puzzles can be solved by the following classification of IC's (See also von Wright [53] for a comparable classification of modalities.)

The constraint $age \in \mathbb{N}$ is an *analytical* statement, i.e. one which follows from the meaning of the symbols occurring in it, and is therefore a necessary truth.[3] This has two important consequences concerning IC change and constraint violation.

1. A *change* of an analytical constraint is a change of the meaning of some of the symbols occurring in it. For example, we may decide to measure temperature in degrees Kelvin instead of degrees centigrade. The constraints $temp \in \mathbb{R}$ then becomes $temp \in \mathbb{R} \wedge temp \geq 0$ (by definition, degrees Kelvin start at 0).[4]
2. An analytical constraint can be used to *check* if the current state of the implementation implements a possible world. If in the implementation a record has a negative value int is age field, the implementation does not represent a possible world. (If the implementation represents a possible world, it *may* represent the correct world.) Thus, an analytical IC constrains the implementation but not the KB since it is necessarily true of the KB.

Consider now the attribute *age* of human beings, measured in years, subject to the constraint that $age \leq 150$. This is not a necessary truth at all but depends upon contingent factors like the state of our general health. It is an *empirical truth* about our UoD which our model, as an abstraction of the UoD, should satisfy. The situation is analogous to empirical laws in natural science. Like the famous Boyle/Carles law for ideal gases $PV/T = C$, the law $age \leq 150$ defines a subset of the total set of logically possible states of the world.

1. Barring change of meaning, a *change* of empirical constraint is a change in the way the UoD behaves.
2. If the implementation violates an empirical constraint, the implementation or the constraint may be in error. The rub is that we (should) choose empirical constraints so weak that any violation can be reasonably be taken to be an implementation error. That is, we only use empirical IC's which during the life of the KB we can reasonably treat *as if* they were necessary truths.[5] That is what usually happens in the literature.

There is a third type of constraint which figures largely in the literature. An example from Nicolas and Yazdanian [42] is the constraint that in a university database, only one teacher teaches a course. This is not an analtyical truth which explicates a meaning relation between "teacher", "teaching" and "course," nor is it an empirical generalization about the way teachers, students and courses happen to behave, but it is a *rule* instituted by agents in the UoD to constrain the possible states of the UoD. We will call these constraints *deontic*. Like empirical constraints, deontic constraints can be violated by the UoD. But where in the case of empirical constraints this is a novel way for the UoD to behave which is logically possible and ethically neutral, in the case of a deontic constraint it is a case of behavior which is logically and physically possible but ethically inadmissible.

1. A *change* of a deontic constraint is a change in the norms pertaining to the UoD. For example, we may decide to allow more than one teacher per course.
2. A deontic constraint can be used to *check* if the agents in the UoD behave in a permissible way. With the constraint in hand, proper action can be taken if two

[3]See Kripke [25, 26] for subtle differences between necessity and analicity. At least the statement that all analytical statements are necessarily true emerges as a necessary truth from that discussion.

[4]We should note here the difficulties of finding a criterion which distinguishes analytical statements, whose truth-value can be determined by an analysis of the meanings of the symbols occurring in it, from empirical statements. Analytical statements are immune to falsification and empirical statements are not. However as Quine argues, immunity to falsification is not a property of a statement in isolation but depends, among other things, on the degree of entrenchment of the statement in a theory. See Davidson [5], Quine [43]. Suppe [49] gives a good summary.

[5]In other approaches, empirical IC's are formulated in a more defeasible way as default knowledge rules. This requires a nonmonotonic logic. We don't open this can of worms here.

teachers are found to teach the same course. Deontic IC's do not constrain the implementation, but the UoD (and hence the KB). The implementation must be able to implement a KB which violates deontic constraints.

Thus, even if we ignore a change of rules governing a UoD, the statement that each course has one teacher is not a necessary truth of a model of a UoD where people can violate the rules. But the statement that there exists the rule that each course *should* have one teacher is a necessary truth in such a model. Of course, if we allow change of rules, the statement may lose its necessary characer. We simplify the treatment of dynamics by considering only change of facts and not change of rules.

We summarize the distinctions between the three types of constraint in the following definition.

Definition 3. An IC is *analytical* iff its truth follows from the (intuitive) meaning of the symbols occurring in it, it is *empirical* iff it states a generalization about the UoD, and it is *deontic* iff it expresses a norm with which the UoD must comply.[6]

We have not distinguished static from dynamic constraints, since the above distinctions are applicable to both types of constraints. We can define the distinction if we view a KB as a set of possible worlds.

Definition 4. An IC is *static* if it is true of each possible UoD state, and it is *dynamic* if at least two states are needed to verify its truth.

Table 1 shows some examples of the different types of constraint.

Remarks.

1. Our example of a dynamic empirical constraint can also be construed deontically as a rule for student behavior. It depends on where the blame is allocated when the rule is violated: with the student violating the rule (so that the rule is deontic) or with the person formulating the rule (so that it is an empirical generalization).
2. It is very hard to find empirical dynamic constraints in social UoD's with the same degree of inviolability as our example of a static empirical constraint. Virtually all empirical generalizations in social science are statistical and can therefore not be used to detect illegal state transitions of a computer.
3. Note that deontic constraints have a strong dynamical flavor, since if state w is inadmissible, then every transition from a permissible state to w is forbidden. For

Table 1.

	Static	Dynamic
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.

[6]The relation between these three classes of statements (and even the validity of the classification) is subject of age-old philosophical controversy. For our pragmatic purposes we can ignore this controversy.

example, because the balance of a bank account may not *be* less than n, any *transition* to such a state is forbidden. Moreover, obligations are wholly dynamical, since they concern actions, not states.

In our view, the KB is an abstract object whose state is determined by 1. the set of abstract objects which exist in that state and 2. the state of each existing object. The KB executes a process wich consists of 1. creation and destruction of abstract objects and 2. the parallel execution of subprocesses by existing objects. Static constraints express truths about all possible KB states, and dynamic constraints specify the process executed by the KB and its objects.[7] This is a view which differs markedly from the view of a KB as a set of sentences.

Our view of a KB as an abstract model of the UoD has uncovered six types of IC's with crucially different characteristics. A simple way to find out how to classify an IC is by allocating the culprit in case of violation. If violation is a mistake of the implementation but cannot occur in the UoD, then it is a necessary constraint. If violation can occur in the UoD and should be corrected in the UoD, then we have a deontic constraint. If violation can occur in the UoD but it is a creative behavior on the part of the UoD, then we have an empirical constraint and the mistake is in our representation of the UoD: we have made a mistake in an empirical generalization about the UoD.

Note that by definition of IC, all IC's are necessary truths. This may seem confusing in the case of deontic constraints, which we have pointed out may be violated. In the next section we clarify this point. For now, we conclude that our language L must contain syntactic constructs which express six different types of IC's. In Chapter 2, we show that a deontic variant of dynamic logic fits these requirements.

We note in passing that the *implementation* of a KB can be in more states than the model it implements and that it can execute more state transitions than are possible in the model. Some of these extra states are needed to implement the model; others are considered to be *error states*. Similarly, some implementation events (or subprocesses) are needed to simulate events in the model; others are erroneous. One of the implementation events must therefore be an *error correction* event, which maps an error state which does not represent a possible world to a state representing a possible world. For example, we need an *undo* event which restores the implementation to a state before an incorrect implementation event occurred. Error correction is not the topic of this paper because by definition the KB itself, as opposed to its implementation, cannot be in error. By contrast, the KB must be able to represent a state which violates a deontic constraint and for each forbidden state it must be able to execute a *violation correction* event which takes it from a forbidden to permissible state. We study such events below.

1.3. *Prescriptive models*

In order to understand deontic IC's, it is essential to understand the relation between a model containing deontic IC's and the UoD. Deontic constraints are not descriptions of but prescriptions for the UoD. The simple picture of a KB as a descriptive model painted in Section 1.1 must therefore be extended to cover the prescriptive aspect of KB's. The distinction between a descriptive and a prescriptive model can be made using the concept of direction of fit, which we borrow, in adapted form, from speech act theory (Searle [46], Searle and Vanderveken [47]).

[7]Just as empirical static constraints are analogous to what are called *laws of coexistence* in the philosophy of science, so empirical dynamic constraints, which describe how abstract objects change their state, are analogous to what are called *laws of change*, like $F = m.a$ (van Fraassen [12]). Because our dynamic constraints also state how the abstract objects composing the abstract KB object interact, empirical dynamic constraints also include *laws of interaction*, which say how entities interact with each other when they are composed into larger systems.

Definition 5. The *direction of fit* between two entities is an arrow which points to the entity to which the other entity must adjust itself in case there is a mismatch between the two. If the direction of fit is from A to B, then B is called *normative* for A.

This definition presupposes that there is a meaningful concept of *matching* between the two entities. In this paper we encounter two such concepts: mismatch between description and described, and mismatch between prescription and the entity whose behavior is prescribed. In each case, the direction of fit is in the direction in which we can find "the boss," which is the entity to whom the other must adjust.

The definition does not intend to exhaust or even approximate the concept of normativity. Explication of this concept is outside the province of computer science.

We can now see that a descriptive model of the UoD has a direction of fit from model to UoD (see Definition 1). The unbroken arrows in figure 1 indicate direction of fit. The direction of fit from T to the model reflects the fact that in our view T, as a theory of the model, must be adapted to changes in the model (which in turn must adapt itself to changes in the UoD).[8] The UoD model is explicated before there is a formal theory which describes it. In KB design, we thus look for IC's which describe a model of the UoD, instead of looking for a model into which we can interpret IC's. The direction of fit between an implementation and a model is from implementation to model, so that each implemented descriptive model plays two roles: it is a *description* of a UoD but a *prescription* for an implementation. The dotted arrows in figure 1 indicate direction of abstraction.

Definition 6. A *prescriptive model* for a UoD is a system whose states and behavior is normative for the UoD.

Note that we speak of a prescriptive model *for* a UoD, in contrast to a descriptive model *of* a UoD. Where descriptive models are used in natural science, which studies systems as they are found in nature, engineering science, which studies the construction of systems which satisfy the demands of a prescriptive model, uses prescriptive as well as descriptive models. The relation between the two kinds of models as used in engineering science is shown in Fig. 2.

Note that M_D is a *descriptive* model of M_P. The distinction corresponds to the two ways of interpreting "It is forbidden to park here," as the *promulgation* of a rule or as the *observation* that a rules exists (von Wright [52], p. 93).

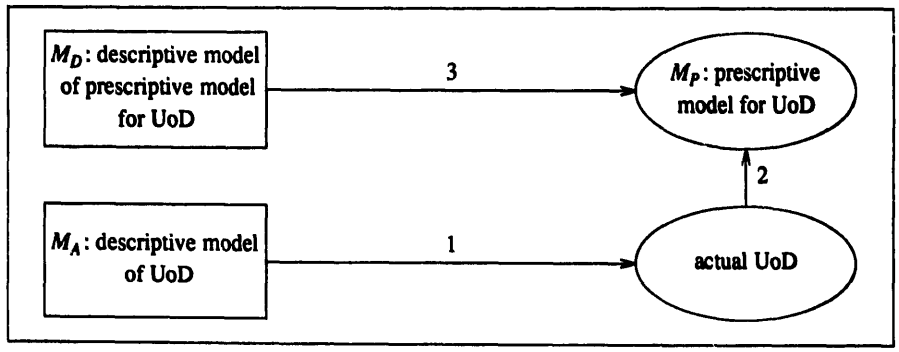

Fig. 2. Prescriptive and descriptive models in engineering science.

[8]This is a simplification in as far as decisions made in the design of L may influence the UoD, and thus the model. There may be a double direction of fit between parts of L and the model. See Section 1.4 for a discussion.

To illustrate the difference between the three models shown in Fig. 2, consider what happens in case of mismatch along each of the three arrows in Fig. 2. If there is a mismatch along arrow 1, M_A is wrong and must be adjusted. If there is a mismatch along arrow 2, we try to adapt the UoD to fit the prescriptive model. But if this turns out to be impossible, or too expensive, or will take too much time, we will adapt our norms, i.e. scale down the prescriptive model so that the demands on the UoD are not so exacting. For example, when we discover during the model-building phase that the cost of living up to all requirements on an elevator system is too high, we may reduce the requirements concerning the quality of elevator service. This makes the situation in engineering science more flexible and therefore more complex: in case of mismatch between prescriptive model and actual UoD, we may adapt either the UoD or the model. This does not affect the direction of fit between the prescriptive model and the actual UoD, as Fig. 3 shows. When we adapt our norms to what is attainable, we replace a prescriptive model by another prescriptive model to which the actual UoD stands in the same direction of fit.

Taking arrow 3 into account, the situation becomes even more complex. In case of mismatch between the behavior of M_A and the actual UoD, we must find our whether we incorrectly modeled the prescriptive model (mismatch along arrow 3) or whether the actual UoD fails to live up to the requirements (mismatch along arrow 2). In the first case, we should improve M_D, in the second we should either improve the actual UoD or scale down M_P.

To illustrate these ideas, take the simulation of queues at counters in a supermarket. If we simulate the queues as they actually occur, the simulation is an implementation of M_A. If we compare this simulation with some required behavior M_P, we formulate this behavior in terms of the available parameters to get M_D and then compare M_D and M_A. What we have not shown in Fig. 3 is that usually we have many M_A's of different possible parameter settings. We then optimize by choosing the best simulation or by scaling down the requirements, if they prove to be too demanding. Note that after we agree on an optimal simulation, the simulated descriptive model M_A then becomes an M_P for the UoD.

The distinction between M_D and M_P allows us to understand why deontic IC's are on the one hand *necessary* statements which, on the other, can be *violated*. A deontic statement, interpreted as a statement about M_P, is a promulgated norm for the behavior of the UoD and as such can be violated by the actual behavior of objects in the UoD. On the other hand,

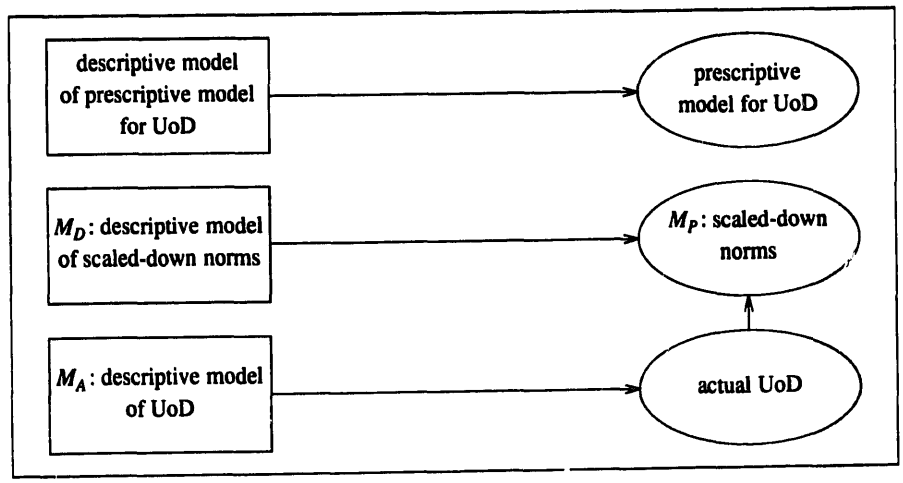

Fig. 3. Adapting the prescriptive model to what is attainable.

interpreted formally as a statement about M_D, it is a necessary truth about a set of possible worlds and cannot be violated by the behavior of objects in those worlds. If it is forbidden to park here, then, ignoring change in the parking rules, it is in all states of the world the case that it is forbidden to park here and as such the rule is necessarily true in all possible states of the world. Nevertheless, actual drivers can violate the rule and park here.

The distinction between M_D and M_P allows us to express a second important fact about deontic IC's. It expresses the fact that M_D is an *abstraction* of M_P. That is, many norms pertaining to the UoD are simply not represented in M_D.

If we want to represent deontic constraints in a KB alongside with descriptive constraints, we get the situation of Fig. 4. An example of this is a library administration, which represents the current state of affairs in its role as M_A ("John has borrowed a book") and represents the required state of affairs in its role as M_D ("John should return the book within 6 weeks").

Insertion of an implementation into the UoD adds complications, as is shown in Fig. 5. We have actually three cases, implementing M_A, implementing M_D, and implementing $M_{A\&D}$. Implementation of M_A alone requires that M_A contain a model of the implementation, because it is a part of the UoD. This part of the descriptive model is usually called a *data dictionary*. Implementing M_A, we can connect the implementation to the rest of the UoD and get a registrative system. To add control, we must also implement part of M_D, in order to have a standard against which to measure actual behavior.

Implementation of M_D alone happens a lot in engineering, for example in the building of airplanes. The designer starts with a list of desiderata (M_P) for actual behavior and models these desiderata in the specification of an airplane. This specification is a model M_D of the desiderata which acts as a prescription for the airplane. Consider now what happens if the plane is built according to specification but does not behave as desired. It may be the case that it does not implement the specification correctly (error along arrow 4), or that the specification does not model the list of desiderata correctly (error along arrow 3) or that the desiderata are unrealistic (M_P is empty). Each of these cases requires a different corrective action. (In reality, of course, the process iterates, usually over scale models or computer models, in order to adjust M_P and M_D to what is possible and desirable.)

Finally, if we implement $M_{A\&D}$ entirely, we have the advantage that in addition to registration we can add a feedback loop for control, but we have the added complexity of the many directions of fit which we get by implementing M_D. An example is a program which controls missiles. Using a model M_A of the missile and input from sensors to the UoD, it compares actual behavior with an implementation of an abstract model M_D of desired

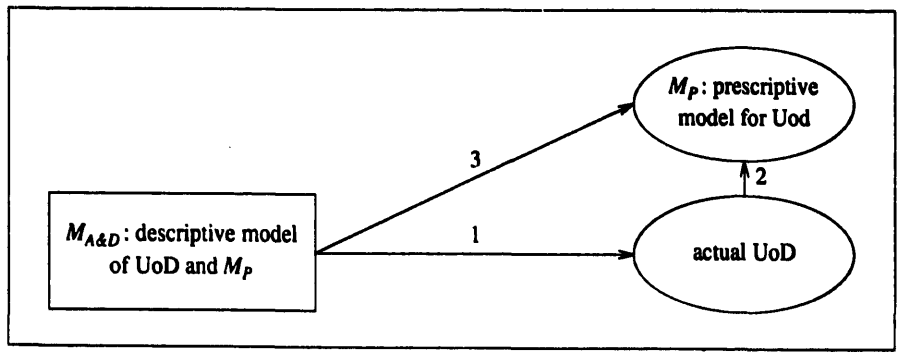

Fig. 4. Merging descriptive models.

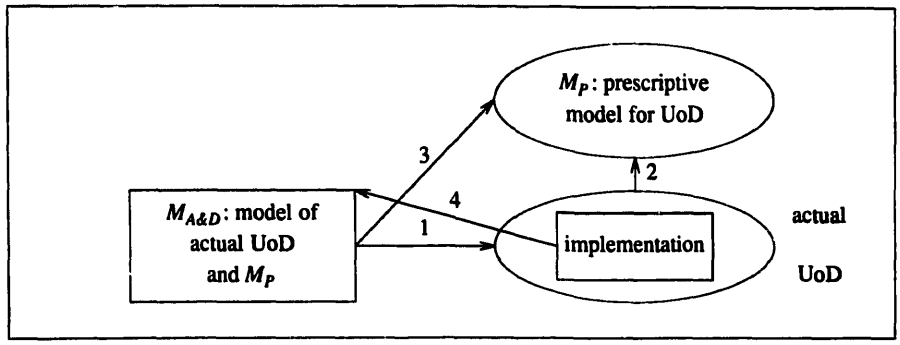

Fig. 5. Implementing and using a model in the UoD.

behavior, and outputs commands to the UoD via effectors to adjust the actual behavior of the UoD. The assumption is, of course, that the mismatch between actual behavior measured by the implementation and the desired behavior as known to the system is along arrow 2 and that the program correctly implements M_A and M_D, which in turn correctly model the UoD and M_D, and M_P is a realistic model for the behavior of the missile.

1.4. *Models of social UoD's*

We have taken examples from social UoD's as well as non-social UoD's. In this section we add a few words about special features of social UoD's. One feature is that we cannot mechanically connect effectors to the people in the UoD in order to control their behavior. People are subject to ethics, and the norms regulating a social UoD have an ethical dimension. This dimension is not captured by the formal logic of norms, which applies to ethical as well as non-ethical norms, but turns up in the actual behavior of the UoD as well as in the degree to which the actual behavior can be influenced. In social UoD's deviations from normal behavior will more frequently occur and will persist longer than in non-social UoD's controlled by a mechanical feedback loop. A KB should thus be able to represent more deviations and should allow for the fact that corrective measures do not have effect, piling deviation upon deviation.

A second feature of social UoD's is that the state of the UoD may be changed simply by representing that change to have taken place. For example, money may be added to a bank account by simply altering a figure in a record in an implementation. There are two ways of looking at this update. In one view, the update mirrors a change which took place in the real world. In another view we make the *performative assumption* that the update *is* a change of the state of the UoD (cf. Lee [28]). The relation between the part of M_D containing the account balance and the UoD is one of *double direction of fit*. The UoD is in a certain state because M_D represents it to be so. This phenomenon occurs frequently in social contexts. For example, a chairperson opens a meeting by declaring it to be opened and authorities close an area for the public by declaring it to be closed. In each case, the agent performing the action causes a state by declaring it to be the case, and in each case, only certain agents can do this, and they must do this in a certain way (cf. Searle and Vanderveken [47], Lehtinen and Lyytinen [29], Auramäki et al. [2]).

Note that making the performative assumption about an action is itself a performative act. A action is performative by declaring it to be so.

The logic of performatives will not be studied in this paper. In Chapter 2 we concentrate on the specification of IC's, in particular deontic IC's.

2. Specification of integrity constraints

2.1. Constraint satisfiability

IC's are sentences (Definition 2) which the KB must satisfy. There are at least two views of satisfaction of an integrity constraint by a KB (Reiter [45]). 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[9]. 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 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 [44], 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 emply models by requiring the KB to be a non-empty model of IC.

Definition 7. For a given logical language L with a class $\mathscr{L}$ 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 $\mathcal{M} \in \mathscr{L}$ such that $\mathcal{M} \models \text{IC}$.
2. The *constraint validation problem* for a KB $\mathcal{M} \in \mathscr{L}$ is the problem of checking whether $\mathcal{M} \models \text{IC}$.

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.

We take it to be part of constraint specification to check whether the constraints are satisfiable. In the next section we show in outline how one can go about showing the satisfiability of a set of constraints. The constraint validation problem, on the other hand, is a problem to be attacked by the implementation of a KB.

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 deduce new information from stored information and can therefore be used to save on storage (Nicolas and Gallaire [41], Nicolas and

[9]Viewing KB as a set of sentences, the consistency view is that KB ∪ IC is satisfiable and the entailment view is that any model of KB must be a model of IC.

Yazdanian [42]). 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.

2.2. Static necessary constraints

Static analytical and empirical 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 function symbols, with metavariables f, g, ... Function symbols with arity 0 are constants. *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 distinguished 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 [44], 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:

$$\forall^E x(\phi(x)) \stackrel{\Delta}{\Leftrightarrow} \forall x(E(x) \Rightarrow \phi(x)) \quad \text{and}$$

$$\exists^E x(\phi(x)) \stackrel{\Delta}{\Leftrightarrow} \exists x(E(x) \wedge \phi(x)) .$$

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 8. A function symbol f of arity $n > 1$ is called *transparent* with respect to a model M of L if for any constants $c_1, \ldots, c_n$, there is a constant c_0 such that $M \models f(c_1, \ldots, c_n) = c_0$.

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.

Definition 9. 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

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

   For an arbitrary closed ϕ, truth in $\mathcal{M}_L$ is defined in the usual way (e.g. see Lloyd [31]).
4. $\mathcal{M}_L$ is a *transparent* Herbrand model of $\mathcal{L}$ if every function symbol of $\mathcal{L}$ is transparent with respect to $\mathcal{M}_L$.

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 [44]). 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.

Our condition on the E predicate in the truth definition plays the role of "meaning postulate" for the existence predicate. If a tuple is in the extension of a predicate, then the constants in the tuple exist.

Definition 10. An *S5 Herbrand–Kripke structure* $\mathcal{K}_L$ of a language L is a collection of transparent Herbrand models of L which are called the *worlds* or *states* of $\mathcal{K}_L$. Truth of a ground atom in $\mathcal{K}_L$ is defined as

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

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 choose an S5 Kripke model because we are only interested in the set of possible worlds and not in a reachability relation on these worlds. Later, we introduce actions with which we can specify state transitions between worlds.

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.

Herbrand–Kripke structures are a particular formalization of our conception of model of Section 1. They function as the integrated prescriptive and descriptive model $M_{A\&D}$ of Figs. 4 and 5 (prescriptive aspects will be added below). Each constant c denotes an object in a possible state of $M_{A\&D}$, 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 11. The *theory* T of $\mathcal{M} \in \mathcal{L}$ is the set of sentences (closed formulas) which are true in $\mathcal{M}$.

Note that a theory of $\mathcal{M}$ is just a set of IC's of $\mathcal{M}$. They are necessary truths of the states of $\mathcal{M}$, so ground atoms, which usually express contingent facts, are usually not part of a theory of $\mathcal{M}$. Our theories are thus different from Reiter's [44] theories, which describe a single state of our model and contain precisely the ground atoms which express the contingent facts of that state.

We now distinguish a number of sets of sentences which are true in models as we have

defined them. Some of these sentences are adaptions of Reiter's [44] well-known closure axioms to Kripke-structures.

Proposition 12. *Given a language L and a model $\mathcal{M} \in \mathcal{L}$ which represents a UoD, a KB theory T of $\mathcal{M}$ consists at least of the following sentences.*

1. *FOL, the set of all theorems of first order predicate logic,*
2. *HB, a set of closure and equality sentences given below,*
3. *D a set of sentences called a domain theory. While FOL and HB are shared by all models of all UoD's, D states truths about a particular model of a particular UoD. The sentences of D are called IC's.*

HB consists of the following sentences.

1. *A domain closure sentence $\forall x(x = c_1 \vee x = c_2 \vee \cdots)$, where all and only the constants of L appear among the c_i.*
2. *Unique name sentence $\neg(c_1 = c_2)$ for all constants c_i and c_j, $i \neq j$.*
3. *Equality sentences*
   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 sentence $\forall \vec{x}, \vec{y}(P(\vec{x}) \wedge x_1 = y_1 \wedge \cdots \wedge x_n = y_n \Rightarrow P(\vec{y}))$, 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 sentence for an n-ary predicate P,*

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

Proof. *FOL* is in any *T*, and the other sentences are true by our Herbrand construction and truth definition. □

Comments:

1. The domain closure sentence says that we chose our objects from the Herbrand universe of *L*, so that in the tuples of a KB we will find any constants from *L*. The unique name sentence says that different names name different objects and the equality sentences state the equivalence properties of =, which follow from the semantics of the = symbol as identity of names. The substitution sentence says that = is a congruence with respect to the predicate symbols and follows from the interpretation of = as identity.
2. Note that the domain closure sentence is infinitely long, so that our language is actually $L_{\omega 1,\omega}$. Cf. Maibaum et al. [32], where the domain closure sentence is called a namability axiom.
3. Reiter [44] 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 sentence 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.
4. If we want to reason about the model we use the derivation rule of *FOL*, modus ponens, as a derivation rule,

$$MP \quad \frac{\vdash \phi, \vdash \phi \Rightarrow \psi}{\vdash \psi}.$$

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. Appendix A shows briefly how this can be done. 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 of theories described in this paper.

Definition 13.

1. If L is L_{Stat} and

   $T = FOL \cup HB \cup D$ with

   $D = Stat$,

   where *Stat* is a set of sentences in L_{Stat}, then T is called a *static* KB theory and D a *static* domain theory. The sentences in *Stat* are called *static IC's*.
2. If L is L_{Dyn} (the language defined in the next section) and

   $T = FOL \cup HB \cup DL \cup D$ with

   DL = the set of axioms of dynamic logic introduced in the next section,

   $D = Stat \cup Dyn$,

   where *Dyn* a non-empty set of sentences in L_{Dyn}, then T is called a *dynamic* KB theory and D a *dynamic* domain theory. The sentences in *Dyn* are called *dynamic IC's*.
3. If L is L_{Deon} (defined in Section 2.4) and

   $T = FOL \cup HB \cup DL \cup D$ with

   $D = Stat \cup Dyn \cup Deon$,

   where *Deon* is a non-empty set of sentences in L_{Deon}, then T is called a *deontic* KB theory and D a *deontic* domain theory. The sentences in *Deon* are called *deontic IC's*.

2.3. Dynamic necessary constraints

We choose a variant of dynamic logic (DL, Harel [16]) 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 (Fiadeiro and Sernadas [11], Sernadas [48]), is too abstract for our purpose. It abstracts from the particular actions executed by a system and instead makes general statements about temporal properties of those actions. We want to be more specific than this and specify particular actions and how they are composed.

For the same reason, temporal logic has a second drawback, which is 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. [8], Lipeck & Saake [30]). 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 strucure 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 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 14. The language L_{Act} of actions, with typical elements α, is given by the following BNF:

$$\alpha ::= a | \alpha_1 \cup \alpha_2 | \alpha_1 \,\&\, \alpha_2 | \bar{\alpha} |\ \textbf{any}\ |\ \textbf{fail}$$

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; $\bar{\alpha}$ 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 the worlds reachable by the execution of arbitrary atomic actions as successor.

For a rigorous semantics of actions we refer to Meyer [35]. (See also appendix C for a synopsis.) For our purpose it suffices to give the following informal definition.

Definition 15. Atomic actions $a \in A$ are interpreted as state transformers in the following sense:

1. Given a state w in a Herbrand–Kripke structure $\mathcal{K}_L$, we associate with each atomic action $a \in A$ a set $W_{a,w}$ of *reachable* states (i.e. Herbrand models) in $\mathcal{K}_L$, which is the set of states that can be reached by performing arbitrary actions that involve a as an element. (More precisely, we consider all subsets of A that contain a. These subsets are "packages" of elementary actions that are performed *simultaneously*.) The set of reachable states is thus the set of worlds which can be reached by performing one step involving an execution of a.

2. $\alpha_1 \cup \alpha_2$ is interpreted as the *set-theoretic union* of the interpretations of α_1 and α_2; this formalizes the meaning of a choice of actions.
3. $\alpha_1 \,\&\, \alpha_2$ is interpreted as the *set-theoretic intersection* of the interpretations of α_1 and α_2; this formalizes the meaning of the simultaneous execution of actions.
4. $\bar{a}$ is interpreted as the set of states that are reachable by the collection of actions which do *not* involve a. (More precisely, we consider all subsets of A that do not contain a.) This formalizes the meaning of $\bar{a}$ as the non-performance of an action.
5. **any** is interpreted as the set of all possible worlds reachable in one step, and **fail** as the empty set.
6. More complex compositions of actions are interpreted as follows: $\overline{\alpha_1 \cup \alpha_2}$ is interpreted as $\bar{\alpha}_1 \,\&\, \bar{\alpha}_2$, $\overline{\alpha_1 \,\&\, \alpha_2}$ as $\bar{\alpha}_1 \cup \bar{\alpha}_2$, and $\bar{\bar{\alpha}}$ as α. Moreover, $\overline{\textbf{any}} = \textbf{fail}$ and $\overline{\textbf{fail}} = \textbf{any}$.

Remarks.

1. Note that the interpretation of $\bar{a}$ is not the set-theoretic complement of the interpretation of a, but the set-theoretic union of interpretations of all one-step actions not containing a.
2. The domain of interpretation of L_{Act} is a Boolean algebra with respect to $\cup$, &, and $^-$, 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 16. The language L_{Trans} of transactions, with typical elements β, is given by the BNF:

$$\beta ::= \alpha \,|\, \beta_1;\, \beta_2 \,|\, \textbf{clock}$$

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 17.

1. Given a state w in a Herbrand-Kripke structure $\mathcal{H}_L$, and a set $W_{\beta_1,w}$ which can be reached from w by β_1, then $\beta_1;\, \beta_2$ is interpreted as set-theoretic union of the sets $W_{\beta_2,w'}$ for $w' \in W_1$.
2. The transaction **clock** is interpreted as

$$\textbf{clock} = \textbf{any}^{n-1};\, inc(t)\,,$$

where n is the number of steps needed to pass one time unit and $inc(t)$ stands for the action which increments the value of the special variable t with 1.

Definition 18. The following abbreviations are used:

$$\beta^n \stackrel{\Delta}{=} \beta;\, \ldots;\, \beta \quad (n \text{ times})$$

$$\alpha_{(n)} \stackrel{\Delta}{=} \textbf{clock}^n;\, \alpha \quad (\text{note: } \alpha_{(0)} \equiv \alpha)$$

$$\alpha_{(\leq d)} \stackrel{\Delta}{=} \alpha_{(0)} \cup \cdots \cup \alpha_{(d)}$$

$$\bar{\alpha}_{(n)} \stackrel{\Delta}{=} \mathbf{clock}^n; \bar{\alpha}$$

$$\alpha_{(>d)} \stackrel{\Delta}{=} \bar{\alpha}_{(\leq d)} = \bar{\alpha}_{(0)} \;\&\; \ldots \;\&\; \bar{\alpha}_{(d)}$$

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

Dynamic constraints

We now extend L_{Stat} to L_{Dyn}. The language we define below is a variant of what is called PDL (Propositional Dynamic Logic) in the literature (Harel [16]).

Definition 19. The language L_{Dyn} of dynamic constraints, with typical elements Φ and Ψ, is given by the BNF:

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

where ϕ is a formula of L_{Stat} and $\alpha \in L_{Act}$.

A dynamic constrain (a formula in L_{Dyn}) is thus a static constraint, or a logical combination of dynamic constraints, or has the form $[\beta]\Phi$ or $DONE\colon \alpha$. $[\beta]\Phi$ 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. $DONE\colon \alpha$ 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 [35], [to appear]. Here, we limit ourselves to the following. Remember that $\mathscr{L}$ is the set of all S5 Herbrand–Kripke structures of L.

Definition 20. Let $w \in \mathscr{K}_L \in \mathscr{L}_{Dyn}$. Then w satisfies $[\beta]\Phi$, written $w \models [\beta]\Phi$, iff

$w' \models \Phi$ for every $w' \in \mathscr{K}_L$ such that the execution of β in w yields w'.

By construction of our class of models $\mathscr{L}_{Dyn}$ and the above truth definition, we know that a number of sentences are true in all models in $\mathscr{L}_{Dyn}$.

Proposition 21. *Given a model $\mathscr{M} \in \mathscr{L}_{Dyn}$, a theory T of $\mathscr{M}$ has the structure $T = FOL \cup HB \cup DL \cup D$, where FOL and HB are as in proposition* 11, *D is a dynamic domain theory and DL is the set*

DL1 $\quad [\alpha](\Phi_1 \Rightarrow \Phi_2) \Rightarrow ([\alpha]\Phi_1 \Rightarrow [\alpha]\Phi_2)$

DL2 $\quad [\alpha_1 \cup \alpha_2]\Phi \Leftrightarrow [\alpha_1]\Phi \wedge [\alpha_2]\Phi$

DL3 $[\alpha_1 \;\&\; \alpha_2]\Phi \Leftrightarrow [\alpha_1](DONE\colon \alpha_2 \Rightarrow \Phi)$

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 $DONE\colon (\alpha_1 \cup \alpha_2) \Leftrightarrow DONE\colon \alpha_1 \vee DONE\colon \alpha_2$

DL9 $DONE\colon (\alpha_1 \;\&\; \alpha_2) \Leftrightarrow DONE\colon \alpha_1 \wedge DONE\colon \alpha_2$

DL10 $DONE\colon \bar{\alpha} \Leftrightarrow \neg\, DONE\colon \alpha$

DL11 $DONE\colon \mathbf{any} \Leftrightarrow \mathbf{true}$

DL12 $DONE\colon \mathbf{fail} \Leftrightarrow \mathbf{false}$

DL13 $[\alpha]DONE\colon \alpha$

DL14 $[\alpha_1]\Phi \Rightarrow [\alpha_2](DONE\colon \alpha_1 \Rightarrow \Phi)$

DL15 $t = n \Rightarrow (\mathbf{clock})t = n + 1$

Proof. Intuitively, this follows from the semantics in Definitions 14 and 16. For a rigorous proof, see Meyer [35]. □

Remarks

1. The axioms are schemata, to be instantiated for each metavariable $\alpha, \beta, \ldots, \Phi$,
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. Note that we do not have an axiom that relates $[\bar{\alpha}]\Phi$ and $[\alpha]\Phi$ in some way. In general there is no such relation. However, DL4 and DL5 indicate how $[\bar{\alpha}]\Phi$ should be evaluated for the non-basic cases in which α is a choice or parallel execution.
5. DL6 is valid because Φ is vacuously true *after* the performance of **fail**, because **fail** has no successor states.
6. 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.
7. 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).
8. If we want to reason about dynamic IC's, we use the derivation rules

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

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.

9. For DL to be true in our model, 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 $DONE$: α, 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 $\quad \forall^E e, s, n(Salary(e, s) \Rightarrow [change - salary(e, n)]Salary(e, s + n))$.

says that for each value of e and n, $change - salary(e, n)$ is a primitive action which has the effect of adding n to the salary of e. Note that the effect of the action is only defined for existing objects.[10] We can add typing information to the axiom as a precondition,

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

Alternatively, we can add the axiom

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

and omit typing information from the dynamic axiom.

2.4. *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: α, one for each of the reasons why the execution of α is forbidden. For each violation predicate, we usually define a *corrective action* which allows one to get out of a state in which that predicate is true.

Definition 22. The language L_{Deon} of deontic constraints is an extension of L_{Dyn} with a distinguished set of predicates, called *violation predicates*, of the form V_i: α, where $\alpha \in A$ and i is a natural number.

Definition 23. The following abbreviations are used.

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

[10]Note also the slight abuse of syntax in that we quantify over action parameters.

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

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

$F(\alpha)$ 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 [1] and has been first formalized in the context of dynamic logic by Meyer [35]. The use of dynamic logic enables the separation of actions from states, which allows one to solve numerous paradoxes of deontic logic (Meyer [34, 35]).

An action is permitted iff it is not forbidden, which is equivalent to saying $P(\alpha) \Leftrightarrow \langle \alpha \rangle \neg V$: α. 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 $[\bar{\alpha}]V$: $\bar{\alpha}$.

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.

An example of a deontic axiom is

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

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 $salary - change(e_1, 1000)$ in state $Salary(e_1, 300)$ leads to a state $Salary(e_1, 1300) \wedge V_1$: $salary - change(e_1, 1000)$. A possible corrective action to this state could be

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

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 $Balance(a, n) \wedge n + m < 0 \Rightarrow [update - balance(a, m)] Balance(a, n)$
A1.2 $Balance(a, n) \wedge n + m \geq 0 \Rightarrow [update - balance(a, m)] Balance(a, n + m)$
or
A2.1 $Balance(a, n) \Rightarrow [update - balance(a, m)] Balance(a, n + m)$
A2.2 $Balance(a, n) \wedge \neg V$: $update - balance(a, m_1) \wedge n + m_2 < 0 \Rightarrow [update -$ balance(a, m_2)] V: update − balance(a, m_2)

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

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

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 $Permissible - overdraw(a, o) \wedge o < n + m_1$ as a precondition and adding A3:

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

$refuse(a, m)$ 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 the 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 [23]). Another choice would have been possible, in which an action is not forbidden because it leads to punishment, 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 true is true, $P\alpha$ or $F\alpha$. Addition of such an assumption would lead to nonmonotonic phenomena (e.g. Etherington [9]).

We now have a formalism to specify models of descriptive as well as prescriptive aspects of the UoD (Fig. 4). In this formalism all deontic formulas are necessary truths, as required by definition 2, but prohibitions can be violated, as required in Section 1.3.

3. Conclusion

In Chapter 1 we distinguished necessary from empirical and deontic constraints. Empirical constraints, if formulated weak enough, can for the purpose of KB design be treated as necessary constraints. Deontic constraints, on the other hand, cannot be treated this way. Violation of a deontic constraint does occur in the UoD and must be represented by the KB. Violation of analytical and suitably chosen empirical constraints is impossible in the KB and, when they occur in the implementation, are errors of the implementation of the KB. Addition of deontic aspects to the KB adds a new dimension to modeling because we now

have descriptive as well as prescriptive aspects in one model. In Chapter 2, 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. [7]). 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.

Dynamic logic has been applied to database specification (as opposed to conceptual modeling of a UoD) by Casanova and Bernstein, who use it to prove properties of data manipulation programs. It has been used by Khosla et al. [24], who use a many-sorted variant of dynamic logic to specify dynamic integrity constraints.

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 [15], 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 [21], Moser [37], Munitz [38]).

Deontic logic is used by Lee [20] 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 [1], Hilpinen [19, 20]), which has been shown by McArthur [33] to contain a number of paradoxes. The deontic variant of dynamic logic which we use does not suffer from these paradoxes (Meyer [34, 35]) 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?

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 an empty 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 play to tackle these questions in the future.

Acknowledgements

We thank the three anonymous referees and Frank Dignum for their constructive criticism of the paper.

References

[1] A.R. Anderson, Some nasty problems in the formalization of ethics, *Noûs* 1 (1967) 345–360.
[2] E. Auramäki, E. Lehtinen and L. Lyytinen, A speech-act-based office modelling approach, *Trans. Office Information Systems* 6, 2 (April 1988) 126–152.
[3] J.A. Bergstra and J.W. Klop, Algebra of communicating processes, in: J.W. de Bakker, M. Hazewinkel and J.K. Lenstra (eds.) *Mathematics and Computer Science I*, CWI Monographs 1 (North-Holland, 1986) 89–138.
[4] J.A. Bergstra and J.W. Klop, Process algebra: Specification and verification in bisimulation semantics, in: M. Hazewinkel, J.K. Lenstra and L.G.L.T. Meertens (eds.) *Mathematics and Computer Science II*, CWI Monographs 4 (North-Holland, 1986) 61–94.
[5] D. Davidson, On the very idea of a conceptual scheme, in D. Davidson, *Inquiries into Truth & Interpretation* (Clarendon Press, 1984) 183–198.
[6] C.L. Chang, DEDUCE 2: Further investigations of deduction in relational databases, in Gallaire and Minker (1978) 201–236.
[7] F. Dignum, T. Kemme, W. Kreuzen, H. Weigand and R.P. van de Riet, Constraint modelling using a conceptual prototyping language, *Data & Knowledge Engineering* 2 (1987), 213–254.
[8] H.-D. Ehrich, E.W. Lipeck and M. Gogolla, Specification, semantics, and enforcement of dynamic database constraints, *Proceedings Tenth Int. Conf. on Very Large Databases*, Singapore (August 1984) 301–308.
[9] D.W. Etherington, *Reasoning with Incomplete Information* (Pitman, 1988).
[10] M.A . Casanova and P.A. Bernstein, A formal system for reasoning about programs accessing a relational database, *Transactions on Database Systems* 2 (1980) 386–414.
[11] J. Fiadeiro and A. Sernadas, Specification and verification of database dynamics, *Acta Informatica* 25 (1988) 625–661.
[12] B.C. van Fraassen, On the extension of Beth's semantics of physical theories, *Philosophy of Science* (1970) 325–339.
[13] H. Freudenthal (ed.), *The Concept and the role of the Model in Mathematics and Natural and Social Sciences* (Reidel, 1961).
[14] H. Gallaire and J. Minker (eds.), *Logic and Databases* (Plenum Press, 1978).
[15] J.J. van Griethuysen (ed.), Concepts and terminology for the conceptual schema and the information base, ISO TC97/SC5/WG3 Report (1982).
[16] D. Harel, Dynamic logic, in: D.M. Gabbay and F. Guenther (eds), *Handbook of Philosophical Logic, Vol. 2* (Reidel, 1984).
[17] R. Harré, *The Principles of Scientific Thinking* (University of Chicago Press, 1970).
[18] M. Hesse, *Models and Analogies in Science* (University of Notre Dame Press, 1966).
[19] R. Hilpinen (ed.), *Deontic Logic: Introductory and Systematic Readings* (Reidel, 1988).
[20] R. Hilpinen (ed.), *New Studies in Deontic Logic* (Reidel, 1988).
[21] J. Hospers, *An Introduction to Philosophical Analysis* (Prentice-Hall, 1953).
[22] G.E. Hughes and M.J. Cresswell, *An Introduction to Modal Logic* (Methuen, 1968).
[23] C.H. Huisjes, *Norms and Logic*, Ph.D. Thesis, Rijksuniversiteit Groningen (1981).
[24] S. Khosla, T.S.E. Maibaum and M. Sadler, Database specification, in T.B. Steel, Jr. and R. Meersman (eds), *Database Semantics (DS-1)* (North-Holland, 1986) 141–158.
[25] S.A. Kripke, Identity and necessity, in S.P. Schwartz (ed.), *Naming, Necessity, and Natural Kinds* (Cornell UP, 1977).
[26] S.A. Kripke, *Naming and Necessity*, revised and enlarged edition (Basil Blackwell, 1980).
[27] C. Kung, A tableaux approach for consistency checking, in A. Sernadas, J. Bubenko and A. Olivé (eds), *Information systems: Theoretical and Formal Aspects* (North-Holland, 1985) 191–207.
[28] R. M. Lee, Bureaucracies as deontic systems, *Trans. Office Information systems* 6, 2 (1988) 87–102.
[29] E. Lehtinen and K. Lyytinen, Action based model of information systems, *Information Systems* 11, 4 (1986) 299–317.
[30] U.W. Lipeck and G. Saake, Monitoring dynamic integrity constraints based on temporal logic, *Information Systems* 12, 3 (1987) 225–269.
[31] J.W. Lloyd, *Foundations of Logic Programming* (Springer, 1984).

[32] T.S.E. Maibaum, P.A.S. Veloso and M.R. Sadler, A theory of abstract data types for program development: Bridging the gap, in H. Ehrich, C. Floyd, M. Nivat and Thatcher (eds) *Proceedings Int. Joint Conf. on Theory and Practice of Software Development (TAPSOFT)*, Springer Lecture Notes in Computer Science 186 (1985) 214–230.
[33] R.P. McArthur, Anderson's deontic logic and relevant implication, *Notre Dame Journal of Symbolic Logic* 22 (1981) 145–154.
[34] J.-J. Ch. Meyer, A simple solution to the "deepest" paradox in deontic logic, *Logique et Analyse* 30 (1987) 81–90.
[35] J.-J. Ch. Meyer, A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic, *Notre Dame Journal of Formal Logic* 19, 1 (1988) 109–136.
[36] J.-J. Ch. Meyer, Using programming concepts in deontic reasoning, to appear in: R. Bartsch, J. van Benthem and P. van Emde Boas (eds), *Semantics and Contextual Expression*, FORIS Publications, Dordrecht-Riverton.
[37] P.K. Moser (ed.), *A Priori Knowledge* (Oxford University Press, 1987).
[38] M.K. Munitz, *Contemporary Analytic Philosophy* (MacMillan, 1981).
[39] E. Nagel, *The Structure of Science* (Harcourt, Brace, 1961).
[40] J.M. Nicolas, Logic for improving integrity checking in relational databases, *Acta Informatica* 18 (1982) 227–253.
[41] J.M. Nicolas and H. Gallaire, Data base: Theory vs. interpretation, in Gallaire and Minker (1978) 33–54.
[42] J.M. Nicolas and K. Yazdanian, Integrity checking in deductive databases, in Gallaire and Minker (1978) 325–344.
[43] W.V.O. Quine, Two dogmas of empiricism, *Philosophical Review* 60 (1951) 20–43.
[44] R. Reiter, Towards a logical reconstruction of relational database theory, in: M. Brodie, J. Mylopoulos and J. Schmidt (eds), *On Conceptual Modelling* (Springer, 1984) 191–233.
[45] R. Reiter, On integrity constraints, in M.Y. Vardi (ed.), *Proc. of the Second Conf. on Theoretical Aspects of Reasoning about Knowledge* (Morgan Kaufmann, 1988) 97–111.
[46] J.R. Searle, *Speech Acts* (Cambridge University Press, 1969).
[47] J.R. Searle and D. Vanderveken, *Foundations of Illocutionary Logic* (Cambridge University Press, 1985).
[48] A. Sernadas, Temporal aspects of logical procedure defintion, *Information Systems* 5 (1980) 167–187.
[49] F. Suppe, *The Structure of Scientic Theories*, 2nd edition (University of Illinois Press, 1977).
[50] P. Suppes, A comparison of the meaning and use of models in mathematics and the empirical sciences, in Freudenthal (1961) 163–177.
[51] R.J. Wieringa and R.P. van de Riet, Algebraic specification of object dynamics in knowledge base domains, in *Proc. 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).
[52] G.H. von Wright, *Norms and Action* (Routledge and Kegan Paul, 1963).
[53] G.H. von Wright, An essay in deontic logic and the general theory of action, *Acta Philosophica Fennica*, Fasc. XXI (North-Holland, 1968).

Appendix A. A supplier KB

This small example is taken from Nicolas [40], who borrowed it from Chang [6]. The UoD consists of a department which buys items from companies and sells them to other parties.

Notational convention: Some unary predicates are set apart as type predicates. For each type predicate, we declare a typical variable and use this variable in the formulas with the understanding that it is of this type. For example, if p is the typical variable of predicate P, then

$\forall p(\phi(p))$ stands for $\forall p(P(p) \Rightarrow \phi(p))$ and

$\exists p(\phi(p))$ stands for $\exists p(P(p) \wedge \phi(p))$.

The IC's are a set of formulas D which are axioms of a domain theory in a language with the following signature.

Signature
Constants:
guns, *bullets*, I, C, T_1, T_2, T_3, T_4, ...
Type predicates: (For each type predicate a typical variable is given.)
Comp(c), c is a company.
Dept(d), d is a department.
Item(i), i is an item.
Type(t), t is the type of an item.
Emp(e), e is an employee.
Other predicates:
$E(x)$, x exists.
Mng(m), m is a manager.
Class(i, t), t is the type of item i.
Subord(e_1, e_2), employee e_1 is a subordinate of employee e_2.

Note that *Mng* is not a type predicate. It is not a type of object but a role an employee object can play. All possible employees are possible managers, but employees are not necessarily created in a world as managers. This difference between *Emp* and *Mng* will become explicit in the dynamic specification. We give two domain theories, one static and one dynamic, which share the following static axioms in *Stat*.

Static necessary constraints
IC0 *Item*(*guns*), *Item*(*bullets*), *Item*(I), *Comp*(C), *Type*(T_1), ...
IC1 $\forall i,\ t(Class(i, t) \Rightarrow Item(i) \wedge Type(t))$
IC2 $\forall m(Mng(m) \Rightarrow Emp(m))$
IC3 $\forall e_1,\ e_2(Subord(e_1, e_2) \Rightarrow Emp(e_1) \wedge Emp(e_2))$
IC4 $\forall^E e_1,\ e_2,\ e_3(Subord(e_1, e_2) \wedge Subord(e_2, e_3) \Rightarrow Subord(e_1, e_3))$

Remarks
1. There are infinitely many constants. In each world, each predicate has a finite extension but since there are infinitely many possible worlds with different constants in the extension of E, there must be infinitely many constants. IC0 therefore is an infinite list of axioms.
2. IC0-IC3 are type axioms. If we would exclude these, we would allow the addition of facts like *Mng*(d) for a department d.
3. IC2 defines the generalization relation between employees and managers.
4. IC4 and 5 explicate the meaning of *Subord*. Note that in IC5 we must restrict quantification to existing objects.

We first give a static domain theory, modeled on Nicolas' solution.

A.1 Static domain theory

We have the following extra predicates.
Supply(c, d, i), company c supplies department d with item i (in the progressive sense, as in "they supply us with bicycle parts.")
Sale(d, i), department d sells item i (as in "shop X sells bicycle parts.")
$V_1(d, i)$, department d sells item i without there being a supplier for i.
$V_2(d, s, i)$, another company than C supplies T_4 items.
$V_3(c)$ company c supplies guns but no bullets.

The following two type axioms for these predicates are to be added as **static necessary constraints** in *Stat*:

IC6 $\forall c, d, i(Supply(c, d, i) \Rightarrow Comp(c) \wedge Dept(d) \wedge Item(i))$
IC7 $\forall d, i(Sale(d, i) \Rightarrow Dept(d) \wedge Item(i))$
The following **static deontic constraints** are to be added to *Deon*:
IC8 If department d sells item i, there should be a supplier s for it:
$\forall^E d, i(\exists^E s(Sale(d, i) \Rightarrow Supply(s, d, i) \Leftrightarrow \neg V_1(d, i)))$
IC9 Only company C supplies type T_4 items:
$\forall^E s, d, i((Supply(s, d, i) \wedge Type(i, T_4) \Rightarrow s = C) \Leftrightarrow \neg V_2(d, s, i))$
IC10 Each company which supplies guns also supplies bullets:
$\forall^E c, d((Supply(c, d, guns) \Rightarrow Supply(c, d, bullets)) \Leftrightarrow \neg V_3(c))$
Nicolas presents these constraints as necessary truths, while they are deontic constraints here, defining when the domain is in a violation state. However, this solution is clearly deficient because we have no way to state how to get out of a violation state, not how we got there in the first place. We now give a dynamic solution to solve this.

A.2 Dynamic domain theory

Interpreting *Supply* and *Sale* as progressive verbs, we must introduce the start and stop of the state introduced by the verb and denote the state itself by the -ing form of the verb.
start-supplying(c, d, i), company c starts to supply department d with item i.
stop-supplying(c, d, i).
start-selling(d, i), department d starts to sell item i.
stop-selling(d, i)
Supplying(c, d, i), company c is supplying department d with item i.
Selling(d, i), department d is selling item i.

In order to be able to give an interesting model of the specification (to show its satisfiability, we can come up with quite trivial models) we introduce events which create objects in the extension of the predicates:
create-comp(c), create a company.
create-dept(d), create a department.
create-item(i, t), create an item of type t.
create-type(t), create a type.
create-emp(e), create an employee.
create-mng(m), create a manager.
become-mng(e), become a manager.
The domain theory now gets a non-empty dynamic part as follows.

Necessary dynamic constraints

Instead of adding IC6 and IC7 to *Stat*, we add the following axioms to *Dyn*:
IC11 $\forall c(Comp(c) \Rightarrow [create\text{-}comp(c)]E(c))$.
IC12 $\forall d(Dept(d) \Rightarrow [create\text{-}dept(d)E(d))$.
IC13 $\forall i, t(Item(i) \wedge Type(t) \Rightarrow [create\text{-}item(i, t)]E(i) \wedge Class(i, t))$.
IC14 $\forall t(Type(t) \Rightarrow [create\text{-}type(t)]E(t))$.
IC15 $\forall e(Emp(e) \Rightarrow [create\text{-}emp(e)]E(e))$.
IC16 $\forall e(Emp(e) \Rightarrow [create\text{-}mng(e)]E(e) \wedge Mng(e))$.
IC17 $\forall^E e(Emp(e) \Rightarrow [become\text{-}mng(e)]Mng(e))$.
IC18 $\forall^E c, d, i(Comp(c) \wedge Dept(d) \wedge Item(i) \Rightarrow [start\text{-}supplying(c, d, i)]Supplying(c, d, i))$
IC19 $\forall^E c,d,i(Comp(c) \wedge Dept(d) \wedge Item(i) \Rightarrow ([start\text{-}supplying(c, d, i)]\neg Supplying(c, d, i))$
IC20 $\forall^E d, i(Dept(d) \wedge Item(i) \wedge E(d) \wedge E(i) \Rightarrow ([Start\text{-}selling(d, i)]selling(d, i))$
IC21 $\forall^E d, i(Dept(d) \wedge Item(i) \Rightarrow ([stop\text{-}selling(d, i)]\neg selling(d, i))$

Remarks.

1. Note that in the creation constraints IC11–IC16 we quantify over all possible objects, whereas in other dynamic constraints we quantify only over existing objects. Events that change a state, only change the state of existing objects.
2. Items are created with a type. Types of items may be created without there being an item of that type.
3. *Mng* is a role of *Emp* objects. Employees can be created manager or can become manager.
4. We did not specify of whom an *m* becomes manager. The predicate *Subord* will thus have no extension in models of our specification, but this can be remedied by adding axioms stating when an employee is a subordinate of another employee.
5. For actions the notational convention to indicate the type of their arguments does not hold. We therefore give the type of definitions of their arguments and preconditions.
6. IC18 and IC19 are the dynamic versions of IC1 and IC20 are dynamic versions of IC3.

Dynamic deontic constraints

The following axioms replace IC8-10 of the static version of *Deon*.

If department d sells item i, there should be a supplier s for it.

IC20 $\forall^E d,\ i(\neg\, Supplying(s, d, i) \Rightarrow F(start\text{-}selling(d, i))$

IC21 $\forall^E d,\ i(V\colon start\text{-}selling(d, i) \Rightarrow [stop\text{-}selling(d, i)]\neg V\colon start\text{-}selling(d, i))$

Only company C supplies type T_4 items:

IC22 $\forall^E c,\ d,\ i(c \neq C \wedge Class(i, T4) \Rightarrow F(start\text{-}supplying(c, d, i))$

IC23 $\forall^E c,\ d,\ i([stop\text{-}supplying(c, d, i)]\neg V\colon start\text{-}supplying(c, d, i))$

Each company which supplies guns also supplies bullets:

IC24 $\forall^E c,\ d([start\text{-}supplying(c, d, guns) \,\&\, \overline{start\text{-}supplying(bullets)}]V\colon start\text{-}supplying\text{-}(c, d, guns))$

IC25 $\forall^E c,\ d(V\colon start\text{-}supplying(c, d, guns) \Rightarrow [start\text{-}supplying(c, d, bullets)]\neg V\colon start\text{-}supplying(c, d, guns))$

IC26 $\forall^E c,\ d(V\colon start\text{-}supplying(c, d, guns) \Rightarrow [stop\text{-}supplying(c, d, guns)]\neg V\colon start\text{-}supplying(c, d, guns))$

IC27–29 Idem with *guns* and *bullets* interchanged

This example shows that in the dynamic specification, we are forced to be more precise than in the static specification, because we must specify how we reach a forbidden state and how we get out of it. In general, a forbidden state arises because an event occurs under specific preconditions. For each event/condition pair, we must remember the combination in the forbidden state in order to be able to get out of the forbidden state in the proper way.

To specify a model for this domain theory, we can start with an initial world w_0 in which the predicates have the following extensions: all type predicates have their constants in their extension and all other predicates have an empty extension. Thus, $[\![Emp]\!]_{w_0} = \{e_1, e_2, \ldots\}$ etc. and $[\![E]\!]_{w_0} = \emptyset$ etc. A Herbrand model containing w_0 as its only world can easily be shown to satisfy all axioms in the domain theory. This is sufficient to show that the domain theory T is satisfiable.

Appendix B. 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: $\{Self, \$2, B1, B_2, \ldots, B_{2000}, P_1, P_2, \ldots, P_{750}, 0, 1, 2, 3, \ldots\}$
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
First-reserver(p, b), p is the first member in the queue of reservations for b
PERF: *borrow*(p, b)p has borrowed the book (and not returned it yet)
Queue(p, b, n), p has number n in the queue of reservers of b
V: α for each of the actions below
Functions:
max(x, y), a function which gives the maximum of two numbers.
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 $X = \{x|P(x)\}$ and $x \in X$ is another way of writing $P(x)$, we use the following abbreviations for cardinality of X in a world,

$$card(X) = \stackrel{\Delta}{\Leftrightarrow} \exists^E_n x \in X .$$

$\exists^E_n x \in X$ means that there are precisely n different elements in X,

$$\exists^E_n x Px \Leftrightarrow$$

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

Assuming that we start with an empty extension for E, are 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), ..., *Book*(B_1), ..., *Natural*(0), ...

IC1 $\forall b\ (Available(b) \Leftrightarrow \forall p, n\ (Present(b) \wedge \neg Queue(p, b, n))$
IC2 $\forall p_1, b\ (First\text{-}reserver(p_1, b) \Leftrightarrow$
$\exists n\ (Queue(p_1, b, n) \wedge \forall p_2, n'(Queue(p_2, b, n') \Rightarrow n' = max(n', n))))$

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 $\forall p, b\ DONE: borrow(p, b) \Rightarrow \overline{PERF: borrow(p, b)}$
IC4 $\forall p, b\ PERF: borrow(p, b) \Rightarrow [\overline{return(p; b)}]PERF: borrow(p, b)$
IC5 $\forall p, b\ [return(p; b)]\neg PERF: borrow(p, b)$

These axioms say that if a *borrow* action has been done, it has been performed (IC3), and that once it has been performed, it remains in the state of having been performed (IC4) until its effect is outdone (IC5). The *return* is a kind of inverse of *borrow*. The behavior of *PERF*: *borrow* contrasts with that of *DONE*: *borrow*, which is only true in worlds resulting from *borrow* and false in other worlds. In principle, inertia axioms like these can be given for each action, but we need only the ones for *borrow*.

IC6 $\forall p, b\ (\neg PERF: borrow(p, b) \Rightarrow [return(p; b)]false)$
IC7 $\forall p, b\ (\neg Present(b) \Rightarrow [borrow(p; b)]false)$
IC8 $\forall p, b, n\ [borrow(p; b)](\neg Queue(p, b, n) \wedge \neg Present(b))$
IC9 $\forall p, b\ [return(p; b)]Present(b)$
IC10 $\forall p, b(\exists n\ (Queue(p_1, b, n) \wedge \forall p_2, n'(Queue(p_2, b, n') \Rightarrow n = max(n', n)))) \Rightarrow$
$[reserve(p; b)]Queue(p, b, n + 1))$
IC11 $\forall p, b, n\ [notify(Self; p, b)][\mathbf{clock}^{(7)}]\neg Queue(p, b, n)$

Remarks.

1. IC6 and IC7 describe necessary preconditions. The other constraints all deal with the effects (postconditions) of actions.
2. According to IC11, a reservation is automatically removed from the queue 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. An implicit assumption of IC11 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

IC12 $\forall p, b_1 P(borrow(p; b_1)) \Leftrightarrow (Member(p) \wedge \neg \exists b_2 V: \overline{return(p, b_2)}$
$\wedge\ card(\{b_3 | PERF: borrow(p, b_3 0)\}) < 3$
$\wedge\ ((Available(b_1) \vee First\text{-}reserver(p, b_1)))$
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 $\forall p, b\ [borrow(p; b)][\mathbf{clock}^{(21)}]$
$(PERF: borrow(p, b) \Rightarrow O(remind(Self; p, b)))$
IC16 $\forall p, b\ (First\text{-}reserver(p, b) \wedge Present(b) \Rightarrow O(notify(Self; p, b)))$
IC17 $\forall p, b\ (V: return(p, b) \Rightarrow O(pay(p; \$2, b)))$
IC18 $\forall p, b\ [borrow(p; b)][\mathbf{clock}^{(21)}](PERF: borrow(p, b) \Rightarrow V: \overline{return(p, b)})$
IC19 $\forall p, b\ [return(p; b)]\neg V: return(p, b)$
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)⇔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 cannot 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 another liability V: *return*(p, b). This liability is cancelled, according to IC20, when the offender pays a fine.
   That he performs a forbidden action when he returns the book too late, follows from $O(\mathit{return}(p; b)_{(\leq 21d)})$ which is equivalent to $F(\mathit{return}(p; b)_{(>21d)})$, and this implies $F(\mathit{return}(p; b)_{(m)})$, for $m > 21d$.
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).

Appendix C. Formal semantics of actions

We give a simplified version of the general semantics in Meyer [35].

Note that A is the set of primitive actions (or rather primitive action *symbols*). We associate with each $a \in A$ a function

$$\rho(a): \mathcal{H}_L \to \mathcal{H}_L$$

describing a's behavior. We write $\rho(A)$ for the set $\{\rho(a) | a \in A\}$ of functions associated with A.

Let $\mathcal{P}^+(\rho(A))$ stand for the collection of finite nonempty sets of $\rho(A)$. The formal semantics of L_{Act} is now given by a semantic function

$$\bar{[\![}\,.\,\bar{]\!]}: L_{Act} \to (\mathcal{H}_L \to \mathcal{P}(\mathcal{H}_L)).$$

In order to define $[\![\,.\,]\!]$, we first define an auxiliary function

$$[\![\,.\,]\!]': L_{Act} \to \mathcal{P}(\mathcal{P}^+(\rho(A)))$$

as follows:

Definition C1.

1. $[\![a]\!]' = \{S \subseteq \rho(A) | \rho(a) \in S\}$.
2. $[\![a_1 \cup \alpha_2]\!]' = [\![\alpha_1]\!]' \cup [\![\alpha_2]\!]'$.

3. $[\![\alpha_1 \ \& \ \alpha_2]\!]' = [\![\alpha_1]\!]' \cap [\![\alpha_2]\!]'$.
4. $[\![\bar{\alpha}]\!]' = \mathcal{P}^+(\rho(A)) \setminus [\![\alpha]\!]'$.
5. $[\![\textbf{fail}]\!]' = \emptyset$.
6. $[\![\textbf{any}]\!]' = \mathcal{P}^+(\rho(A))$.

Remarks.

1. expresses that the semantics of a is defined by all simultaneously performed "packages" that contain its (i.e. a's) behavior. Likewise 4. implies that the semantics of $\bar{a}$ is determined by all simultaneously performed "packages" that do *not* contain a's behavior.

Next we define an auxiliary function

$$R: \mathcal{P}^+(\rho(A)) \to (\mathcal{H}_L \to \mathcal{P}(\mathcal{H}_L))$$

as follows:

Definition C2.
Let $S = \{\rho(a_1), \ldots, \rho)(a_n)\} \subseteq \rho(A)$, $n \geq 1$. Then

$R(S) = \{(\rho(a_1) \circ \cdots \circ \rho(a_n))(w)\}$ if $\rho(a_1), \ldots, \rho(a_n)$ are compatible for argument w,

$R(S) = \emptyset$ otherwise,

where functions $f_1, \ldots, f_n$: $\mathcal{H}_L \to \mathcal{H}_L$ are *compatible* for w if $(f_1 \circ \cdots \circ f_n)(w) = (f_{i_1} \circ \cdots \circ f_n)(w)$ for any permutation $(i_1, \ldots, i_n)$ of $(1, \ldots, n)$.

R is lifted to the domain $\mathcal{P}(\mathcal{P}^+(\rho(A))$ in the usual way: let $T \subseteq \mathcal{P}^+(\rho(A))$. Then $R(T)(w) = \cup_{S \in T} R(S)(w)$.

Now we define our semantic function $[\![\,.\,]\!]$: $L_{Act} \to (\mathcal{H}_L \to \mathcal{P}(\mathcal{H}_L))$ by

$$[\![\alpha]\!] = \lambda w \,.\, R([\![\alpha]\!]')(w) \,.$$