Annals of Mathematics and Artificial Intelligence 3 (1991) 393–428

THE INHERITANCE OF DYNAMIC AND DEONTIC INTEGRITY CONSTRAINTS

or:

Does the boss have more rights?

R.J. WIERINGA, H. WEIGAND *, J.-J.Ch. MEYER **
and F.P.M. DIGNUM +

Vrije Universiteit, Amsterdam, The Netherlands

Abstract

In [18,23], we presented a language for the specification of static, dynamic and deontic integrity constraints (IC's) for conceptual models (CM's). An important problem not discussed in that paper is how IC's are inherited in a taxonomic network of types. For example, if students are permitted to perform certain actions under certain preconditions, must we repeat these preconditions when specializing this action for the subtype of graduate students, or are they inherited, and if so, how? For static constraints, this problem is relatively trivial, but for dynamic and deontic constraints, it will turn out that it contains numerous pitfalls, caused by the fact that common sense supplies presuppositions about the structure of IC inheritance that are not warranted by logic. In this paper, we unravel some of these presuppositions and show how to avoid the pitfalls. We first formulate a number of general theorems about the inheritance of necessary and/or sufficient conditions and show that for upward inheritance, a closure assumption is needed. We apply this to dynamic and deontic IC's, where conditions are *pre*conditions of actions, and show that our common sense is sometimes mistaken about the logical implications of what we have specified. We also show the connection of necessary and sufficient preconditions of actions with the specification of weakest preconditions in programming logic. Finally, we argue that information analysts usually assume "constraint completion" in the specification of (pre)conditions analogous to predicate completion in Prolog and circumscription in non-monotonic logic. The results are illustrated with numerous examples and compared with other approaches in the literature.

1. Introduction

1.1. THE PROBLEM OF DYNAMIC CONSTRAINT INHERITANCE

The specification of dynamic constraint inheritance in a taxonomy contains some pitfalls that one must be very careful to avoid. For example, Borgida et al. [3] give the following dynamic constraints (we simplify the first constraint a bit).

* Now at Tilburg University, Tilburg, The Netherlands.
** Also, University of Nijmegen, Nijmegen, The Netherlands.
+ Now at the University of Swaziland.

1. A student can enroll in a course if the course is not full.
2. Undergraduate students must in addition have a permission if they want to take a graduate course.

Adding the static constraints

3. An undergraduate student is a student,
4. A graduate course is a course

then from

John is an undergraduate trying to enroll without permission in a graduate course that is not full,

we can infer the following:

by 3 and 4 he is a student trying to enroll without permission in a course that is not full,

so by 1 he is permitted to enroll, for students can enroll in a course which is not full.

This is obviously not the intention of the designers, who mean rule 2 to apply whenever an undergraduate takes a graduate course. What has gone wrong can be explained if we formalize the constraints in dynamic logic. Informally, read $[\alpha]\phi$ in the following examples as

"action α necessarily leads to a state where ϕ holds."

Then it is tempting to formalize the constraints as follows.

$$\forall c, s(Course(c) \wedge \neg full(c) \wedge Student(c) \Rightarrow [enroll(c, s)]enrolled(c, s)) \quad (1)$$

$$\forall c, s\ (GradCourse(c) \wedge \neg full(c) \wedge Undergrad(s) \wedge permission(c, s) \Rightarrow [enroll(c, s)]enrolled(c, s)) \quad (2)$$

$$\forall s(Undergrad(s) \Rightarrow Student\ (s)) \quad (3)$$

$$\forall c(GradCourse(c) \Rightarrow Course(c)). \quad (4)$$

The formalized inference is now:

$$Undergrad(s) \wedge GradCourse(c) \wedge \neg full(c) \wedge \neg permission(c, s) \vdash$$

$$Student(s) \wedge Course(c) \wedge \neg full(c) \wedge \neg permission(c, s) \vdash$$

$$[enroll(c, s)]enrolled(c, s) \wedge \neg permission(c, s).$$

The problem with the formalization (1)–(4) is that in (1)–(2), the arrows point the wrong way. (1) says that it is a *sufficient* condition for enrollment that the course is not full. But then (2) cannot be relevant for any enrollment for which the sufficient condition in (1) is satisfied. An improved formalization, which blocks the fallacious inference above, says that the course not being full is a *necessary* condition for enrollment of a student, but it leaves open whether there may be other necessary conditions as well. So we get

$$\forall c, s(Course(c) \wedge Student(s) \wedge [enroll(c, s)]enrolled(c, s) \Rightarrow \neg full(c)) \quad (1')$$

$$\forall c, s(GradCourse(c) \wedge Undergrad\ (s) \wedge [enroll(c, s)]enrolled(c, s) \Rightarrow permission(c, s)). \quad (2')$$

(1′) says that a necessary condition for enrollment is that the course be not full, but leaves open that there are other conditions. (2′) gives one of these other conditions, for a special case of (1′). In section 5 below we will show that this allows us to infer

$$\forall c, s(GradCourse(c) \wedge Undergrad(s) \wedge [enroll(c, s)]enrolled(c, s) \Rightarrow \neg full(c) \wedge permission(c, s)). \quad (3')$$

At least three other problems can be pointed out in this example. First, (2) is really meant as an *exception* to the general rule under (1). This is not captured by the formalization above, and in general leads to a special (non-monotonic) logic. In section 6.2, we show how exceptions can be specified in a monotonic logic.

Second, one of the reasons that (1) looks more natural than (1′) is that we tend to read a *temporal order* into the implication sign. This is unjustified, for the only temporal ordering in the formulas above is implied by the modal operator $[\alpha]$ for an action α.

Third, although we talked informally about being allowed to enroll in a course, we formalized the postcondition *enrolled*(c, s) explicitly. In database applications, we often are only interested in an action being *allowed* by the rules and regulations of the universe of discourse (UoD), or, weaker, of an action being *possible* in the current state of the UoD, independently of specifying what the exact result of the action is. In this paper, we offer a logic to specify these modalities. We do this in a deontic variant of dynamic logic introduced by Meyer [16,17] and applied to conceptual model (CM) specification in two earlier papers [18,23]. Deontic logic is a logic of norms and is excellently suited to specifying what is allowed in a UoD. Dynamic logic specifies what can happen and what is the result of an action, and can be used to specify dynamic integrity constraints (IC's). We first give some necessary background that motivates the application of deontic logic to CM specification.

1.2. THE UoD, CONCEPTUAL MODELS, AND INTEGRITY CONSTRAINTS

The CM is an abstraction of a UoD, and is at the same time a mathematical structure into which a theory (specification) *Spec* is interpreted, as shown in fig. 1. In section 2 we define a CM as a Kripke structure consisting of a set of *possible worlds* without an explicit accessibility relation. A possible world, also

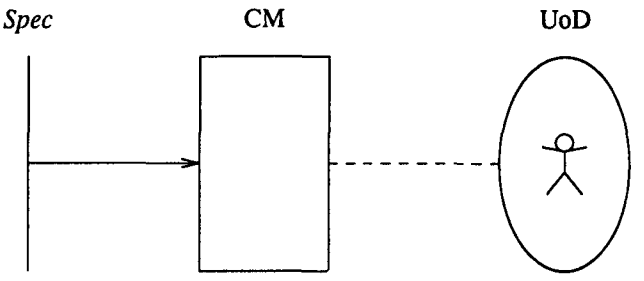

Fig. 1. The double role of a CM.

called a *state* of CM, abstractly represents a possible state of the UoD. Accessibility between worlds is not defined explicitly by a binary relation on possible worlds, but by *a set of actions*, which are functions on the possible worlds of the model. Details are given below. Here we want to make two claims about the relation between IC's and the UoD.

1. Explicit knowledge about the UoD is always expressed in closed formulas in which the variables range over classes of possible objects in the UoD.
2. All IC's express explicit knowledge about the UoD.

This means that IC's are closed sentences in the language of *Spec* that are true of the CM. So they are axioms of *Spec*, or logical consequences of them. But then IC's are *necessary truths* of the CM, for they are true in all states of CM. This means that knowledge about what is *usually* the case in the UoD (empirical knowledge), or about what *ought* to be the case in the UoD (deontic, i.e. normative knowledge), cannot be added to *Spec* in the same way as genuine necessary truths can. A statement like

$age \geqslant 0$

is an analytic truth, i.e. it follows from the meaning of the symbols occurring in it. It is therefore also a necessary truth and can, translated to the appropriate language, be adopted in *Spec*. But assuming we are talking about the age of persons, a statement like

$age < 100$

is an empirical truth which may be violated by the UoD and can therefore not be added as a necessary truth to *Spec* without further qualification. Similarly, talking about bank accounts,

$balance \geqslant 0$

is not a necessary truth either but prescribes the way UoD entities should behave.

What we have now is, on the one hand, a formalism and a semantics which allow us to express only necessary truths as IC's, and on the other two types of knowledge, empirical and normative, which allow exceptions and cannot be expressed in a straightforward way as necessary truths. Our solution to the problem for empirical constraints is to simply formulate them so widely that we do not, in the useful life of the CM, encounter any exceptions. This in itself is an empirical prediction which may be falsified, but in this paper we ignore problems arising from this kind of falsification. For deontic constraints, we retreat to a kind of metalevel by stating the fact that there is a norm as a necessary truth valid for all states of the CM. We then provide a mechanism by which deviations from this norm in a particular world can be detected and, most importantly, corrected.

Returning to fig. 1, *Spec* is roughly analogous to a database (DB) schema in that it specifies all possible states of a CM. The CM as a whole can thus be represented in a computer by storing *Spec*. We do not assume anything about

how a particular state of the CM is represented in the computer. Traditionally, relational DB's store a set of ground atoms; in the presence of *Spec*, though, these allow us to infer many other facts not explicitly stored. Alternatively, we can store a set of sentences which has the represented CM state as its preferred model. Our goal in this paper is not to discuss the relative merits of different ways to represent CM states, but to discuss the relative merits of certain ways to specify, in *Spec*, necessary truths about all states of the CM.

1.3. STRUCTURE OF THE PAPER

In section 2 we give a brief introduction to the language L_{Deon} defined in [23]. Section 3 then introduces types, so that in section 4 we can introduce IC's as closed sentences in which all variables are quantified over types. We give a classification of IC's with respect to whether they specify necessary or sufficient conditions for static formulas, or for actions, or for deontic modalities. Section 5 then contains the main result of the paper, concerning the inheritance of necessary or sufficient conditions in a taxonomy. Due to the generality of this result, it is applicable to a large number of quite complex constraints. This is illustrated with a number of examples. In section 6 we compare this with some non-monotonic approaches and draw some methodological conclusions from this. In particular, we show how constraints can be "completed" in a way roughly analogous to predicate completion in Prolog and to circumscription in AI approaches. Section 7 contains a summary of the main conclusions that can be drawn from the paper.

2. Syntax and semantics of L_{Deon}

Our specification language consists of three parts, L_{Stat} for static integrity constraints, L_{Dyn} for dynamic constraints, and L_{Deon} for deontic constraints. We devote a brief section to each of these parts.

2.1. THE STATIC LANGUAGE L_{Stat}

L_{Stat} is a simple first-order language with the following syntax.

- Examples of variables are p, b, $\cdots$. The letters x, y and z (possibly indexed) are always used as metavariables over the variables. There are infinitely many variables.
- Constants are $A101$, 1234,... and the letter c (possibly indexed) is used as metavariable over the constants. There are infinitely many constants.
- There are finitely many function symbols, with metavariables f, g, $\cdots$.
- There are finitely many 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).

Terms and formulas are built in the usual way using $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\forall$, $\exists$, and punctuation symbols (,), [and]. We use infix notation for $=$. Metavariable over terms is t and metavariables over formulas are ϕ and ψ. The existence predicate E is used by convention to single out the set existing objects among the set of possible objects. The following abbreviations are used:

$\forall^E x[\phi(x)] := \forall x[E(x) \Rightarrow \phi(x)]$ and

$\exists^E x[\phi(x)] := \exists x[E(x) \wedge \phi(x)]$.

In the following definition, we presuppose the usual model concept from first-order predicate logic.

DEFINITION 2.1

A function symbol f of arity $n > 1$ is called transparent with respect to a structure M of L_{Stat} 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 2.2

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* $\mathscr{B}_L$ over the universe U_L of L is the set of all ground atoms (closed atomic formulas) of L containing no function symbols.
3. A *Herbrand structure* $\mathscr{M}_L$ is a subset $\mathscr{M}_L \subseteq \mathscr{B}_L$. Truth in $\mathscr{M}_L$ is defined for ground atoms as

   $\mathscr{M}_L \models P(c_1, \cdots, c_n) := P(c_1, \cdots, c_n) \in \mathscr{M}_L$

   and $E(c_i) \in \mathscr{M}_L$, $i = 1, \cdots, n$.

   For an arbitrary closed ϕ, truth in $\mathscr{M}_L$ is defined in the usual way (e.g. see [13]).
4. $\mathscr{M}_L$ is called a *transparent* Herbrand structure of L if every function symbol of L is transparent with respect to $\mathscr{M}_L$.

Note that the use of transparent Herbrand structures is a technical convenience that could be eliminated. Without transparent Herbrand structures, the construction would be more difficult due to the presence of the equality symbol.

DEFINITION 2.3

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

$\mathscr{PW}_L \models \phi \Leftrightarrow w \models \phi$ for all $w \in \mathscr{PW}_L$.

Truth of a closed formula ϕ in a single transparent Herbrand structure $w \in \mathscr{PW}_L$ is defined in the usual way. The collection of all Herbrand-Kripke structures of L is called $\mathscr{L}$.

We will drop the qualification "S5" from the definition from now on. In terms of fig. 1, a CM is a Herbrand-Kripke structure and *Spec* is a theory of the CM. Note that in general, theories have no unique model. We just assume that there is an *intended* model of *Spec*. Static integrity constraints are then sentences in L_{Stat} that are true in the CM. An example of a static constraint is

$$\forall b(bird(b) \Rightarrow warmblooded(b)) \tag{5}$$

2.2. THE DYNAMIC LANGUAGE L_{Dyn}

L_{Stat} is extended to a variant of dynamic logic L_{Dyn}. (See [8] for dynamic logic.) This is done by adding languages for actions and transactions, which are defined first. For the *action language* L_{Act} we assume a fixed set A of *atomic actions*.

DEFINITION 2.4

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

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

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

The semantics of actions is that they are functions on $\mathscr{PW}_L$. A formal semantics is given in [17], which is summarized in [23]. Note that if an action changes the world (if it is not the identity function), it does so instantaneously, i.e. there are no intermediate worlds during the execution of an action. The execution of an action is also called a *step*. The action **fail** has no successor worlds and the action **any** executed in a world has an arbitrary world as successor, which may be the world in which it is executed.

DEFINITION 2.5

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

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

where $\alpha \in L_{Act}$.

Intuitively, $\beta_1;\beta_2$ is the sequential composition of the transactions β_1 and β_2. In the formal semantics, sequential composition of actions is interpreted as a string of atomic actions.

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 to match the duration of one time unit.

DEFINITION 2.6

The following abbreviations are used:

$\beta^n := \beta; \cdots ; \beta$ (n times)

$\alpha_{(n)} := \textbf{clock}^n;\ \alpha$ (note: $\alpha_{(0)} \equiv \alpha$)

$\alpha_{(\leqslant d)} := \alpha_{(0)} \cup \cdots \cup \alpha_{(d)}$

$\bar{\alpha}_{(n)} := \textbf{clock}^n;\ \bar{\alpha}$

$\alpha_{>d} := \bar{\alpha}_{(\leqslant d)} \equiv \bar{\alpha}_{(0)}\ \&\ \cdots\ \&\ \bar{\alpha}_{(d)}$

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

DEFINITION 2.7

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

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

where ϕ is a formula of L_{Stat}. We use

$\langle\beta\rangle\Phi$

as an abbreviation of $\neg[\beta]\neg\Phi$.

The intuitive semantics of $[\alpha]\phi$ is that after the execution of α, ϕ necessarily holds. $[\alpha]\phi$ itself is thus the *weakest precondition of* α *with respect to the*

postcondition ϕ. If $[\alpha]\phi$ holds in a world and α is executed in that world, then afterwards ϕ will hold.

Just like [.] is a necessity operator, $\langle . \rangle$ is a possibility operator. $\langle \alpha \rangle \phi$ says that doing α *may* lead to a state where ϕ, i.e. there is at least one world in which doing α leads to a world where ϕ.

An example of a dynamic logic expression is

$$\forall e(\neg \mathit{Employee}(e) \Rightarrow [\mathit{hire}(e)]\, \mathit{Employee}(e)). \tag{6}$$

This says that not being an employee is a sufficient condition for $\mathit{hire}(e)$ to necessarily lead to a state where e is an employee.

Note that we misuse notation slightly by parametrizing actions with variables that also occur in the predicates. We have not provided for this in the syntax or semantics of the language. However, this can easily be rectified, see for example [7].

DEFINITION 2.8

The following *objective modalities* are introduced by definition:

- $\mathbf{POS}(\alpha) := \neg[\alpha]\mathbf{false}$ ("α can possibly happen"),
- $\mathbf{NEC}(\alpha) := [\bar{\alpha}]\mathbf{false}$ ("α necessarily happens"),
- $\mathbf{DIS}(\alpha) := \neg\mathbf{NEC}(\alpha)$ ("α is discretionary, may not happen"), and
- $\mathbf{IMP}(\alpha) := \neg\mathbf{POS}(\alpha)$ ("α can impossibly happen").

These are called *objective* modalities to distinguish them from the deontic ones which are introduced below. Both are dynamic in that they apply to actions, but where objective modalities state concern what objectively *can* happen, deontic ones concern what is *admissible*. We have

$$\mathbf{NEC}(\alpha) \Leftrightarrow \neg\mathbf{POS}(\bar{\alpha}), \quad \mathbf{DIS}(\alpha) \Leftrightarrow \mathbf{POS}(\bar{\alpha}), \quad \text{and} \quad \mathbf{IMP}(\alpha) \Leftrightarrow [\alpha]\mathbf{false}.$$

A simple example is

$$\forall p,\ b(\mathit{Person}(p) \wedge \mathit{Book}(b) \Rightarrow \mathbf{POS}(\mathit{borrow}(p, b)), \tag{7}$$

which can be paraphrased as "It is possible for a person to borrow a book" (of course, other entities, like institutions, may also be able to borrow books).

2.3. THE DEONTIC LANGUAGE L_{Deon}

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

out of that state. The necessary reductions are then effected by the following definition.

DEFINITION 2.9

The following abbreviations are used for deontic modalities:
- $\mathbf{P}(\alpha) := \neg[\alpha]V_i\colon \alpha$ for an i ("α is permitted"),
- $\mathbf{O}(\alpha) := [\bar{\alpha}]V_i\colon \alpha$ for an i ("α is obligatory"),
- $\mathbf{D}(\alpha) := \neg\mathbf{O}(\alpha)$ ("α is discretionary, not obligatory"), and
- $\mathbf{F}(\alpha) := \neg P(\alpha)$ ("α is forbidden").

Note the analogy with dynamic modalities. The following equivalences are analogous to those for objective modalities.

$\mathbf{O}(\alpha) \Leftrightarrow \neg\mathbf{P}(\bar{\alpha})$, $\mathbf{D}(\alpha) \Leftrightarrow \mathbf{P}(\bar{\alpha})$, and $\mathbf{F}(\alpha) \Leftrightarrow [\alpha]V_i\colon \alpha$ for an i.

The following interesting implications follow immediately from the definitions:

$\mathbf{P}(\alpha) \Rightarrow \mathbf{POS}(\alpha)$, $\mathbf{NEC}(\alpha) \Rightarrow \mathbf{O}(\alpha)$, $\mathbf{IMP}(\alpha) \Rightarrow \mathbf{F}(\alpha)$, and $\mathbf{D}(\alpha) \Rightarrow \mathbf{DIS}(\alpha)$.

The first of these says that what is permitted, is possible. This is a lot weaker than the Kantian doctrine "Ought implies can." [1] The sentence $\mathbf{O}(\alpha) \Rightarrow \mathbf{POS}(\alpha)$ is *not* provable in L_{Deon}. Neither is, incidentally, $\mathbf{O}(\alpha(x) \Rightarrow \mathbf{P}(\alpha(x))$ provable in L_{Deon}.

An example of a deontic constraint is

$$\forall p, b\big(Person(p) \wedge Book(b) \Rightarrow [borrow(p, b)]\mathbf{O}(return(p, b)_{(\leqslant 30)})\big) \qquad (8)$$

The meaning of this is that if a person borrows a book, then (afterwards) he must return it at most 30 days later. Note that it is not guaranteed that he returns it. Contrast this with

$$\forall p, b\big(Person(p) \wedge Book(b) \Rightarrow [borrow(p, b)]\mathbf{NEC}(return(p, b)_{(\leqslant 30)})\big), \qquad (9)$$

which can be paraphrased as "if a person borrows a book, then afterwards he necessarily returns it". As a statement about the UoD, this is patently false, and as an IC for the DB, failure to return a book within 30 days would cause the DB to be in an inconsistent state. The DB would still be consistent, though, when (8) were used instead of (9). Moreover, we can specify what should happen when a person does not return a book:

$$\forall p, b(V\colon return(p, b) \Rightarrow \mathbf{O}(pay(p, \$2, b))). \qquad (10)$$

When the book is finally returned, the violation is undone,

$$\forall p, b[return(p, b)]\neg V\colon return(p, b) \qquad (11)$$

but the fine must still be paid.

[1] Cf. S. Körner, *Kant* (Penguin, 1955), "What ought to be must be possible, since every moral obligation implies the (moral or *noumenal*) freedom to realize it." (p. 167.) The Kantian doctrine is a rationalization of one of the central tenets of Lutheranism. The converse is the Modern Engineer's doctrine "Can implies ought", or, freely translated, "If you can do something, you must try it out."

3. Natural kinds and roles

If knowledge is expressed as closed statements about objects of a certain type, as pointed out in section 1.2, then we must be able to talk about types. In this section we add the concept of a type to L_{Deon}. We follow Sowa [22] in using an explicit *type* predicate to declare the type of a term.

DEFINITION 3.1

Let T be a finite set of constants not occurring in L_{Deon}. The elements of T are called *type names* and τ is used as metavariable over T. L_{Deon} is extended to the typed language TL_{Deon} as follows.

1. TL_{Deon} contains a special binary predicate *type* and the set T of type names. The only well-formed atomic formulas that can be built with *type* are of the form $type(t, \tau)$ for a term t and a type name τ, and the only place where τ can occur is as the second argument of *type*. $type(x, \tau)$ is called a *declaration* of x.
2. We introduce the abbreviations

   $\forall x: \tau(\phi(x)) := \forall x(type(x, \tau) \Rightarrow \phi(x))$ and

   $\exists x: \tau(\phi(x)) := \exists x(type(x, \tau) \wedge \phi(x))$.
3. The language TL_{Deon} is the set of all *closed* statements that can be built this way and which have all their variables typed. The inference relation $\vdash$ is defined as usual for first-order logic. We only consider formulas in prenex normal form, i.e. $Q_1x_1 \cdots Q_nx_n(\phi(x_1, \cdots, x_n))$, where $x_1, \cdots, x_n$ are all the free variables in ϕ and Q_i are quantifiers. Because all variables are typed, we can write this as

   $Q_1x_1: \tau_1 \cdots Q_nx_n: \tau_n(\phi'(x_1, \cdots, x_n))$,

   with $\tau_i \in T$.

So far, we have defined a syntax of a first-order language containing some special predicates like *type* and E, and a distinguished set T of constants. We must now give a semantics to the type names. We have a choice of keeping the extension of a type name constant in each world, or varying it. This choice has an intuitive meaning, for compare the types *Person* and *Employee*. Some objects can become employees or cease to be employees without coming into existence or ceasing to be. There is life before being hired by a company, as well as after terminating a contract. On the other hand, there is no kind of object that can become a person without coming into existence, or that can cease to be a person without ceasing to exist. Apparently, being a person is an essential property of objects in the way that being an employee isn't. We will call types like *Person* *natural kinds* and types like *Employee roles*. With this, we have sufficient motivation for the following definition.

DEFINITION 3.2

1. We assume that T is partitioned into the sets $\mathscr{K}$ and $\mathscr{R}$. The constants in $\mathscr{K}$ will be called *natural kind names* and those in $\mathscr{R}$ *natural role names.* Metavariable over $\mathscr{K}$ is k and over $\mathscr{R}$ is r. τ is still our general metavariable over $\mathscr{K} \cup \mathscr{R}$.
2. A *typed structure* $\mathscr{PW}_T L_{Deon}$ of TL_{Deon} consists of a structure $\mathscr{PW}_{L_{Deon}}$ of the untyped version L_{Deon} of TL_{Deon}, and assignments

   $\| . \|_w : \mathscr{K} \to \left(\wp(U_{L_{Deon}})\right)$ for each world $w \in \mathscr{PW}_{L_{Deon}}$, and

   $\| . \|_w : \mathscr{R} \to \left(\wp(U_{L_{Deon}} \cap \| E \|_w)\right)$ for each world $w \in \mathscr{PW}_{L_{Deon}}$,

   where $U_{L_{Deon}}$ is the universe of $\mathscr{PW}_{L_{Deon}}$, and we must have:
   - for each $r \in \mathscr{R}$ there are worlds w, $w' \in \mathscr{PW}_{TL_{Deon}}$ with $w \neq w'$ and $\| r \|_w \neq \| r \|_{w'}$,
   - for each $k \in \mathscr{K}$ and all w, $w' \in \mathscr{PW}_{TL_{Deon}}$ with $w \neq w'$, we have $\| k \|_w = \| k \|_{w'}$, and
   - $U_{L_{Deon}} = \bigcup_{\tau \in \mathscr{K}} \| \tau \|$.

   The elements of $\| \tau \|$ are called the possible *instances* of τ.
3. Truth for formulas *type* (c, τ) in $w \in \mathscr{PW}_{TL_{Deon}}$ is defined by

   $w \vDash type(c, \tau) :\Leftrightarrow c \in \| \tau \|_w$.

   Truth in $\mathscr{PW}_{TL_{Deon}}$ is defined as usual.

We drop the index TL_{Deon} from $\mathscr{PW}_{TL_{Deon}}$ when the language is clear or can be presupposed to be clear.

Remarks

(1) The extension of $k \in \mathscr{K}$ in w is independent of w. So when t has a natural kind, it has that kind in all possible worlds. This formalizes the intuition that being a member of a natural kind belongs to the *essence* of an object, where the essence of an object can be defined as the underlying structure of the object. (Cf. Kripke [11,12] and Putnam [19] discuss the concept of a natural kind, Wieringa [24] applies this to object-oriented conceptual modeling.)

(2) Our semantics requires every constant to be in the extension of at least one natural kind.

(3) The extension of a role name in a world must fall within the extension of the existence predicate in that world. This formalizes the intuition that $\| t \|_w \in \| r \|_w$ if t denotes an object actually playing role r. Whereas a role is something an object has in a certain context, a natural kind is an underlying structure of an object independent of context.

(4) Sort names in many-sorted logic denote natural kinds, not roles. Thus, it is wrong to formalize *Student*, *Employee* etc. as sort names.

Note that since our logic is not many-sorted, we need not define the argument sorts of predicate symbols and the argument and result sorts of function symbols. Thus, our syntax is unsorted. However, we must now require from every theory in TL_{Deon} that there are type axioms for all predicate and function symbols that it uses. These must be of the form

$$P(x_1, \cdots, x_n) \Rightarrow type(x_1, \tau_1) \wedge \cdots \wedge type(x_n, \tau_n).$$

So where many-sorted logic gives the argument and result sorts in the signature of the theory, we must give them in the theory itself.

TL_{Deon} has the advantage over many-sorted logic of being able to say that the type of an object changes. Assuming that $Person \in \mathscr{K}$ and $Employee \in \mathscr{R}$, we can specify

$$\forall x(type(x, Person) \Leftrightarrow [hire(x)]type(x, Employee)). \tag{12}$$

Since an object cannot change its natural kind, we cannot consistently specify

$$\forall x(type(x, Person) \Leftrightarrow [die(x)]\neg type(x, Person)). \tag{13}$$

4. Specifying necessary and sufficient conditions

We can now express knowledge about classes of objects. In order to be clear about the role of necessary and sufficient conditions in the expression of knowledge, we give here a classification of IC's with respect to this aspect. All IC's we consider in this section are of the form

$$\forall x : \tau(\Phi(x) \Rightarrow \Psi(x))$$

with $\Phi(x), \Psi(x) \in TL_{Deon}$.

DEFINITION 4.1

1. In

   $$\forall x : \tau(\Phi(x) \Rightarrow \Psi(x)), \tag{14}$$

   $\Phi(x)$ is called a *sufficient condition* of $\Psi(x)$ for objects of type τ, and $\Psi(x)$ a *necessary condition* of $\Phi(x)$ for objects of type τ.
2. In

   $$\forall x : \tau(\Phi(x) \Rightarrow [\alpha(x)]\Psi(x)), \tag{15}$$

   $\Phi(x)$ is called a *sufficient precondition of* $\alpha(x)$ *with respect to* $\Psi(x)$ for objects of type τ.
3. In

   $$\forall x : \tau([\alpha(x)]\Phi(x) \Rightarrow \Psi(x)), \tag{16}$$

   $\Psi(x)$ is called a *necessary precondition of* $\alpha(x)$ *with respect to* $\Phi(x)$ for objects of type τ.

There is no concept of "sufficient postcondition". Postconditions are always necessary, but this is opposed to *possibility,* whereas necessary preconditions are opposed to sufficient preconditions. Thus, ϕ in $[\alpha]\phi$ is opposed to ψ in $\langle\alpha\rangle\psi$, whereas $[\alpha]\phi \Rightarrow \psi$ is opposed to $\psi \Rightarrow [\alpha]\phi$. For example, selling a thing necessarily changes ownership, but dropping it may possibly break it:

$$\forall p_1, p_2: \mathit{Person}, t: \mathit{Thing}([\mathit{sell}(p_1, t, p_2)]\mathit{own}(p_2, t)) \tag{17}$$

$$\forall t: \mathit{Thing}(\langle \mathit{drop}(t)\rangle \mathit{broken}(t)). \tag{18}$$

Whenever $\Phi \Rightarrow \Psi$, we call Φ *stronger* than Ψ and Ψ *weaker* than Φ. The strongest statement is **false** (it implies everything) and the weakest statement is **true** (it is implied by everything). Thus, a sufficient precondition of $[\alpha(x)]\Psi(x)$, such as $\Phi(x)$ in (15), is stronger than $[\alpha(x)]\Psi(x)$ itself. If we weaken $\Phi(x)$ in (15) until it is equivalent with $[\alpha(x)]\Psi(x)$, then it is the *weakest precondition* of $[\alpha(x)]\Psi(x)$. So any statement equivalent to $[\alpha(x)]\Psi(x)$ is a weakest precondition of $\alpha(x)$ with respect to $\Psi(x)$. In particular, $[\alpha(x)]\Psi(x)$ itself is a weakest precondition of $\alpha(x)$ with respect to $\Psi(x)$. Modulo logical equivalence, one can speak of *the* weakest precondition of $\alpha(x)$ with respect to $\Psi(x)$ [1].

On the other hand, all necessary preconditions of $[\alpha(x)]\Phi(x)$ are weaker than it. If we strengthen $\Psi(x)$ in (16) until it is equivalent with $[\alpha(x)]\Phi(x)$, then it is a weakest precondition of $\alpha(x)$ with respect to $\Phi(x)$. Thus, if $\Psi(x)$ is the weakest precondition of $\alpha(x)$ with respect to $\Phi(x)$, we must read this as

> "$\Psi(x)$ is the weakest precondition that must hold so that $\alpha(x)$ necessarily leads to $\Phi(x)$; any weaker precondition does not guarantee that $\Phi(x)$ holds after $\alpha(x)$."

There are five interesting types of instances of the general IC form, according to whether Φ and Ψ are static or dynamic, and for the second case whether they express objective or deontic modalities.

4.1. STATIC CONSTRAINTS

These are of the form

$\forall x: \tau(\phi(x) \Rightarrow \psi(x))$ ϕ is sufficient for ψ and ψ is necessary for ϕ.

for $\phi(x)$, $\psi(x) \in L_{Stat}$. An example is the following improvement of (5):

$$\forall x: \mathit{Animal}(\mathit{type}(x, \mathit{Bird}) \Rightarrow \mathit{warmblooded}(x)). \tag{19}$$

This says that for an animal, being warm-blooded is a necessary condition for being a bird, and that for animals, being a bird is sufficient for being warm-blooded.

4.2. DYNAMIC CONSTRAINTS

4.2.1. Postconditions

We are only interested in necessary postconditions, not in possible postconditions. An important form of this (with $\Phi \equiv$ **true**) is

$\forall x : \tau([\alpha(x)]\phi(x))$ α leads necessarily to ϕ.

for $\phi(x) \in L_{Stat}$. (We do not consider nested postconditions such as $[\alpha][\beta]\phi$.) We saw already sufficiently many examples of this. An interesting subcase is that of **role-changes.** These are of the form

$\forall x : k([\alpha(x)]\, type(x, r) \wedge E(x))$.

For example,

$$\forall^E p : Person([register(p)]\, type(p, Student)) \tag{20}$$

Note that we require p to exist, so that existence need not be explicitly mentioned as postcondition. The statement is ill-formed if $Student \notin \mathscr{R}$.

4.2.2. Preconditions

We are interested in necessary as well as sufficient preconditions. For $\phi(x)$, $\psi(x) \in L_{Stat}$, these have the form

$\forall x : \tau(\phi(x) \Rightarrow [\alpha(x)]\psi(x))$	If $\phi(x)$, then $\alpha(x)$ necessarily leads to $\psi(x)$. $\phi(x)$ is a sufficient precondition for $\alpha(x)$ to lead necessarily to $\psi(x)$.
$\forall x : \tau(([\alpha(x)]\psi(x)) \Rightarrow \phi(x))$	If $\alpha(x)$ necessarily leads to $\psi(x)$, then $\phi(x)$. $\phi(x)$ is a necessary precondition for $\alpha(x)$ to lead necessarily to $\psi(x)$.

For example, when the queue of book reservations has n elements, then after reserving a book, it has $n + 1$ elements:

$$\forall n : integer(queue(n) \Rightarrow [reserve\_book]\ queue(n + 1)) \tag{21}$$

When after a reservation the queue has $n + 1$ elements, then before the reservation it has n elements:

$$\forall n : integer([reserve\_book]\, queue(n + 1)) \Rightarrow queue(n) \tag{22}$$

A negative form of sufficient preconditions gives us the form of frame axioms:

$\forall x : \tau(\neg\phi(x) \Rightarrow [\overline{\alpha(x)}]\neg\phi(x))$.

For example

$$\forall p_1, p_2 : Person(\neg married(p_1, p_2) \Rightarrow [\overline{marry(p_1, p_2)}]\neg married(p_1, p_2)), \tag{23}$$

which says that marrying is the only way to get married. In other words, the state of being married is invariant under any action but marrying.

Preconditions for objective modalities have the form

$\forall x: \tau(\mathbf{POS}(\alpha(x)) \Rightarrow \phi(x))$	$\phi(x)$ is a necessary precondition for $\alpha(x)$ to be possible.
$\forall x: \tau(\phi(x) \Rightarrow \mathbf{POS}(\alpha(x)))$	$\phi(x)$ is a sufficient precondition for $\alpha(x)$ to be possible.

Similar schemas exist for the other objective modalities.

For example, someone can only sell something, if he owns it:

$$\forall p: \mathit{Person}, t: \mathit{Thing}(\mathbf{POS}\ (\mathit{sell}(p, t)) \Rightarrow \mathit{own}(p, t)). \tag{24}$$

If you are a person, you can borrow a book:

$$\forall p: \mathit{Person}, b: \mathit{Book}(\mathbf{POS}(\mathit{borrow}(p, b)). \tag{25}$$

These are the preconditions that are often used in database research.

Preconditions for deontic modalities have the form

$\forall x: \tau(\mathbf{P}(\alpha(x)) \Rightarrow \phi(x))$	α is allowed to occur only if ϕ holds.
$\forall x: \tau(\phi(x) \Rightarrow \mathbf{P}(\alpha(x)))$	If ϕ holds, α is allowed to occur.

Similar schemas exist for the other deontic modalities.

For example,

$$\forall p_1, p_2, p_3: \mathit{Person}, t: \mathit{Thing}(\mathbf{P}(\mathit{sell}(p_1, t, p_2)) \Rightarrow \neg\exists^E p_3: \mathit{promised}(p_1, t, p_3) \wedge \neg p_2 = p_3, \tag{26}$$

which can be paraphrased as "Someone can sell something only if he has not promised it to someone else." Finally, "if someone is a manager, he or she is permitted to park"

$$\forall m(\mathit{type}(m, \mathit{Manager}) \Rightarrow \mathbf{P}(\mathit{park}(m)). \tag{27}$$

5. Inheritance of constraints

DEFINITION 5.1

TL_{Deon} is extended to a language OTL_{Deon} with ordered types by adding a distinguished binary infix predicate $\leqslant$. The only well-formed atomic formulas that can be built with $\leqslant$ are of the form $\tau_1 \leqslant \tau_2$ for $\tau_1, \tau_2 \in \mathrm{T}$. The semantics of $\leqslant$ is:

$$\tau_1 \leqslant \tau_2 := \forall x(\mathit{type}(x, \tau_1) \Rightarrow \mathit{type}(x, \tau_2)).$$

THEOREM 5.2

1. $\leqslant$ is a partial ordering on T.
2. If $\tau_1 \leqslant \tau_2$, then $\| \tau_1 \|_w \subseteq \| \tau_2 \|_w$.
3. $\| k \|_w = \| k \|_{w'}$ for all $w, w' \in \mathscr{P}\mathscr{W}_L$.

4. If for each role name r, there is a possible world $w \in \mathscr{PW}_{L_{Deon}}$ with $\| r \|_w = \emptyset$ and for each natural kind name k, there is a world w with $\| k \|_w \neq \emptyset$, then if $k \leqslant \tau$, then $\tau \in \mathscr{K}$.
5. $k_1 \leqslant k_2$ iff there is a $w \in \mathscr{PW}$ with $\| k_1 \|_w \subseteq \| k_2 \|_w$.
6. $r_1 \leqslant r_2$ iff $\| r_1 \|_w \subseteq \| r_2 \|_w$ for all $w \in \mathscr{PW}$.

Proof

1, 2 and 3 are trivial.

For 4, note that $k \leqslant \tau$ implies $\forall x(type(x, k) \Rightarrow type(x, \tau))$, so

$\| k \|_w \subseteq \| \tau \|_w$ for all $w \in \mathscr{PW}$.

There is a world with $\| k \|_w \neq \emptyset$, and because $k \in \mathscr{K}$, by 3 we have $\| k \|_w \neq \emptyset$ for all w. So $\| \tau \|_w \neq \emptyset$ for all $w \in \mathscr{PW}$. Now, if τ would be a role, then there is a w_0 with $\| \tau \|_{w_0} = \emptyset$, and we would have

$\emptyset \neq \| k \|_{w_0} \subseteq \| \tau \|_{w_0} = \emptyset$,

which is a contradiction.

5 is trivial, and 6 follows from the definition of $\leqslant$ and the truth definition. □

Remarks

(1) Under the weak assumptions that for each role there is at least a world where it is not played, and that each natural kind has a non-empty extension, we have that no role can be larger than a natural kind. This has a practical consequence for the type hierarchy specified by the information analyst, for it excludes certain taxonomic structures.

(2) 4 and 5 give a way how to verify, in the UoD, whether the ordering on type names is correct. The condition on the relative ordering on role names is quite strong. In general, if there is a world with $\| r_1 \|_w \subseteq \| r_2 \|_w$, we need not have $r_1 \leqslant r_2$. For example, if in the current world all students are employees, we need not have *Student* $\leqslant$ *Employee*. On the other hand, if for natural kinds *Car* and *Vehicle* we have $\| Car \|_w \subseteq \| Vehicle \|_w$ in at least one world, then *Car* $\leqslant$ *Vehicle*.

DEFINITION 5.3

Let *Spec* be a specification in OTL_{Deon}. The partially ordered set (poset) (T, $\leqslant$) specified by *Spec* is called the *taxonomy* specified by a specification in OTL_{Deon}. If $\tau_1 \leqslant \tau_2$, then τ_1 is called a *specialization* of τ_2 and τ_2 a *generalization* of τ_1.

5.1. DOWNWARD INHERITANCE

The basic property of taxonomies is of course that properties inherit downwards. We call this *basic inheritance* and state it in the following theorem.

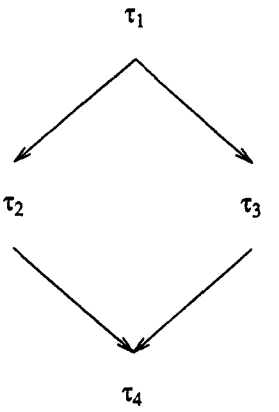

Fig. 2. Inheritance relations.

THEOREM 5.4

$\tau_2 \leqslant \tau_1 \Rightarrow (\forall x : \tau_1\Phi(x) \Rightarrow \forall x : \tau_2\Phi(x))$.

Proof

Trivial. □

We are generally interested in situations as in fig. 2, where τ_1 has several specializations and τ_4 has several generalizations. Downward inheritance is inheritance of properties from τ_1 to types smaller than τ_1. Basic inheritance says that the number of properties increase as the types get smaller. To determine thoughts, we fix a few concepts.

DEFINITION 5.5

The inheritance of constraints from a single source is called *single inheritance*, and inheritance from multiple sources *multiple inheritance*. Inheritance in the direction of the arrows is called *downward* and inheritance against the direction of the arrows *upward.*

In fig. 2, there is single downward inheritance from τ_1 to τ_2 and τ_3, and single upward inheritance from τ_4 to τ_2 and τ_3. There is multiple upward inheritance from τ_2 and τ_3 to τ_1, and multiple downward inheritance from τ_2 and τ_3 to τ_4.

In this section we look at the two possible cases of downward inheritance for our standard form of constraints $\forall x : \tau(\Phi_\tau(x) \Rightarrow \Psi_\tau(x))$.

Single downward inheritance from τ_1 to τ_2 and τ_3 is trivial and is described by the basic inheritance property. Multiple downward inheritance of necessary or sufficient conditions is a bit less trivial but follows straightforwardly from basic inheritance.

COROLLARY 5.6

In figure 2, if

$\forall x: \tau_2(\Phi_2(x) \Rightarrow \Psi(x))$ and $\forall x: \tau_3(\Phi_3(x) \Rightarrow \Psi(x))$

then

$\forall x: \tau_4(\Phi_2(x) \vee \Phi_3(x) \Rightarrow \Psi(x))$,

and if

$\forall x: \tau_2(\Phi(x) \Rightarrow \Psi_2(x))$ and $\forall x: \tau_3(\Phi(x) \Rightarrow \Psi_3(x))$

then

$\forall x: \tau_4(\Phi(x) \Rightarrow \Psi_2(x) \wedge \Psi_3(x))$.

Proof

By basic inheritance, in the first case we have

$\forall x: \tau_4((\Phi_2(x) \Rightarrow \Psi(x))) \wedge (\forall x: \tau_4(\Phi_3(x) \Rightarrow \Psi(x)))$.

By the laws of predicate logic, this is equivalent to

$\forall x: \tau_4(\Phi_2(x) \vee \Phi_3(x) \Rightarrow \Psi(x))$,

and similarly for the second case. □

We can summarize this by the slogan

necessary conditions inherit downwards conjunctively, and sufficient conditions inherit downwards disjunctively.

This implies that necessary conditions become stronger as we go down in the taxonomy, and sufficient conditions become weaker. In particular, necessary *pre*-conditions of actions become stronger, and sufficient preconditions weaker, as we go to more specialized types. This is the reason why in the student example mentioned in the introduction, where we want the precondition for enrollment for more specialized types to be stronger, we must specify necessary and not sufficient preconditions. This example is formalized at the end of the following section to illustrate upward inheritance. Here we give some examples of corollary 5.6, following the classification of IC's given in section 4.

For **static constraints**, take $StudEmp \leqslant Student$ and $StudEmp \leqslant Emp$, and

$$\forall s: Student,\ n: Natural(age(s, n) \Rightarrow n > 18) \tag{28}$$

$$\forall e: Emp,\ n: Natural(age(s, n) \Rightarrow n > 21). \tag{29}$$

Then the conjunction of these necessary conditions hold for student employees. Assuming some natural axioms for $>$ on *Natural*, we get

$$\forall se: StudEmp,\ n: Natural(age(se, n) \Rightarrow n > 21). \tag{30}$$

For **postconditions**, the theorem implies that the effects of an action accumulate as we go down the taxonomy. For example, suppose all persons have a salary and all students and employees are persons. Let *salary* be defined in the axioms as a function with argument sort *Person* and result sort *Natural*. Now let salary increases be defined somewhat disadvantageous for students:

$$\forall e: Emp,\ n: Natural\ (salary(e,n) \Rightarrow [inc\text{-}salary(e)]\, salary(e) = n + 10) \tag{31}$$

$$\forall s: Student,\ n: Natural\ (salary(s,n) \Rightarrow [inc\text{-}salary(s)]\, salary(s) = n - 10). \tag{32}$$

If there are student employees, we get

$$\forall se: StudentEmp,\ n: Natural\ (salary(se) = n$$
$$\Rightarrow [inc\text{-}salary(se)]\, salary(se) = n + 10$$
$$\wedge\, [inc\text{-}salary(se)]\, salary(se) = n - 10), \tag{33}$$

which implies

$$\forall se: EmpStudent,\ n: Natural\ (salary(se) = n \Rightarrow [inc\text{-}salary(se)]\ salary(se)$$
$$= n + 10 \wedge salary(se) = n - 10),$$

which, assuming the usual axioms for natural numbers, implies

$$\forall se: EmpStudent,\ n: Natural(salary(se,n) \Rightarrow [inc\text{-}salary(se)]\mathbf{false}).$$

The result is that according to the specification, an attempt to execute *inc-salary* for student employees deadlocks. This mistake in the specification could have been avoided if common constraints are specified as high up in the taxonomic hierarchy as possible. Because *salary* is a person attribute, the constraint should have been specified there:

$$\forall p: Person,\ n: Natural\ (salary(p,\ n) \Rightarrow [inc\text{-}salary(p)]\, salary(p) = n + 10). \tag{34}$$

If exceptions must be made to this general rule, then this can be done by defining appropriate disjoint subtypes of person. We illustrate this in section 6.3.

We can of course monotonically add more effects of an action when we specialize. Suppose each person has an *age*, which is increased by the *inc-age* action for all persons. For employees, an employer wants to specify as extra effect of *inc-age* a bonus salary increase. This can be specified as follows:

$$\forall e: Emp,\ n: Natural\ (salary(e,\ n) \Rightarrow [inc\text{-}age(e)]\, salary(e) = n + 10). \tag{35}$$

If this is the only extra effect specified for *inc-age*, then it is not derivable that

$$\forall s: Students,\ n: Natural\ (salary(s,\ n) \Rightarrow [inc\text{-}age(s)]\, salary(s) = n + 10). \tag{36}$$

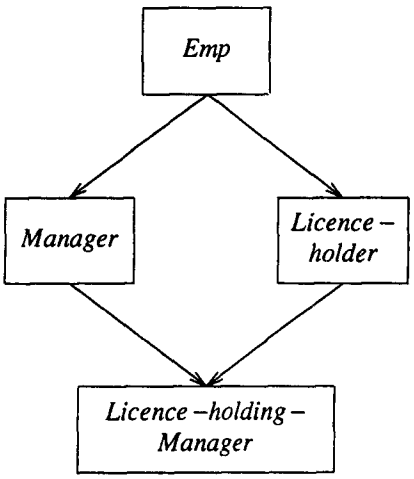

Fig. 3. Employee taxonomy.

We can also redeem our promise delivered in the introduction to show that (1′) and (2′) implies (3′). We have that, with *Undergrad* $\leqslant$ *Student* and *GradCourse* $\leqslant$ *Course*, (1′) implies

$$\forall c: \mathit{GradCourse},\ s: \mathit{Undergrad}\ [\mathit{enroll}(c, s)]\,\mathit{enrolled}(c, s) \Rightarrow \neg \mathit{full}(c), \qquad (1'')$$

which gives (3′) with (2′).

As an example concerning **deontic modalities**, we will try to find out whether the boss has more rights, as asked in the subtitle of this paper. We assume *Boss* $\leqslant$ *Emp*. We then have

$$\forall e: \mathit{Emp}\ (\mathbf{P}\,\alpha(e)) \Rightarrow \forall b: \mathit{Boss}\ (\mathbf{P}\alpha(b)), \qquad (37)$$

so that the boss seems to have more permissions than the average employee. On the other hand, the boss has also more obligations, for

$$\forall e: \mathit{Emp}\ (\mathbf{O}\alpha(e)) \Rightarrow \forall b: \mathit{Boss}\ (\mathbf{O}\alpha(b)), \qquad (38)$$

which may be a comfort to some.

Looking at the multiple downward inheritance of preconditions, the picture becomes more complicated. In fig. 3, we show a taxonomy of employees, managers, and licence holders. Licence holders are employees who have a licence to park their car on numbered parking lots. Managers, on the other hand, have more rights. For them, permission to park on any lot is granted if the lot is free. These constraints are illustrated in fig. 4, where we omitted the quantifications $\forall m$: *Manager*, $\forall lot$: *ParkingLot*, and $\forall lh$: *Licence-holder*. Clearly, multiple downward inheritance gives us an unwanted implication, as illustrated in fig. 4. This is not inconsistent in itself, but it would be inconsistent with any fact representing a particular lot to be not full and unnumbered.

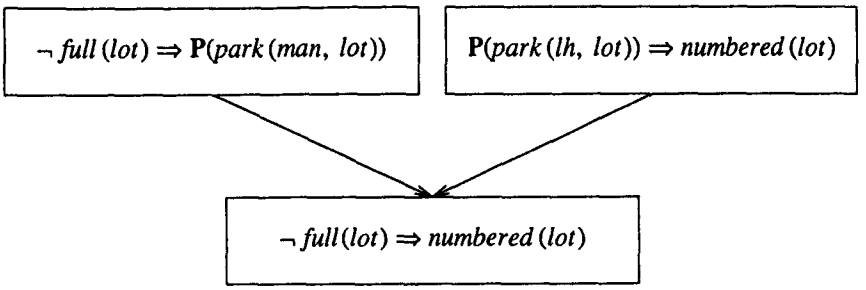

Fig. 4. Inconsistent multiple downward inheritance.

Before we improve this specification, note the following implication of the constraint for licence holders:

$$\mathbf{P}(\,park(lh,lot)) \Rightarrow numbered(lot) \vdash \neg numbered(lot) \Rightarrow \mathbf{F}(\,park(lh,lot)), \quad (39)$$

where we again dropped the quantifications. A licence holder is not permitted to park on any unnumbered lot. This is reasonable, but may not be what we think we specified with the **P**-form of the constraint.

The source of the problem in fig. 4 is that the sufficient condition of $\mathbf{P}(\,park(man,\ lot))$ interacts in an unwanted way with the necessary condition for $\mathbf{P}(\,park(lh,\ lot))$. The more logical way to do this is to specify $\mathbf{P}(\,park(man,\ lot)) \Rightarrow \neg full(lot)$. Put this way, it is just a logical necessary precondition of any park action that the lot be empty. In accordance with the rule that constraints should be specified as high up in the taxonomy as possible, we get fig. 5.

As a final illustration of how multiple downward inheritance can accumulate to unwanted constraints, consider figs. 6 and 7. If factory workers are permitted

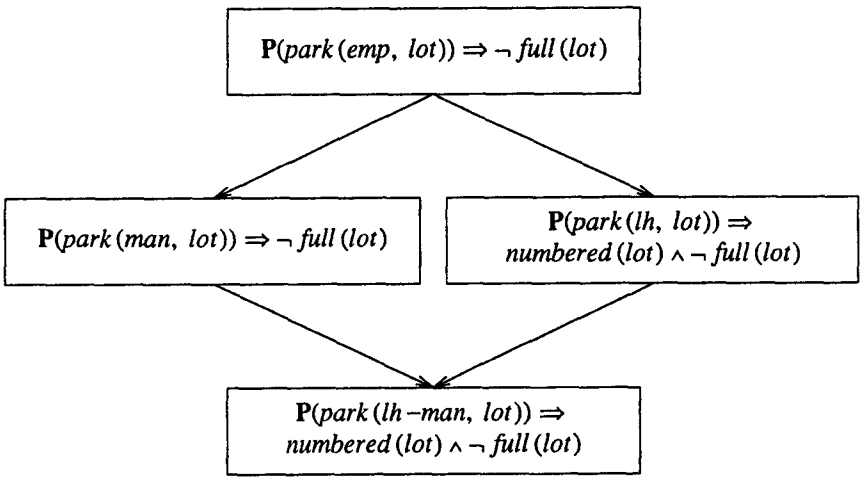

Fig. 5. Improved specification.

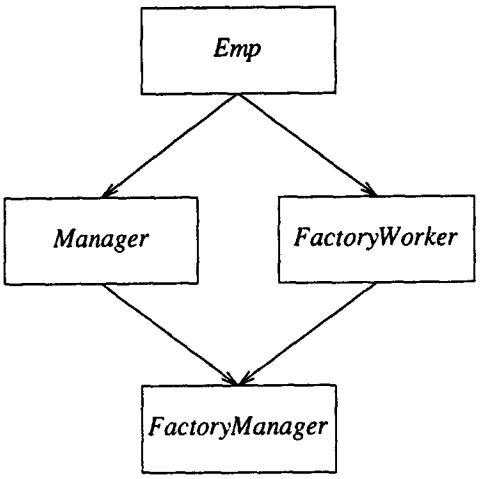

Fig. 6. Another employee taxonomy.

to park on unnumbered lots, and managers are permitted to park on numbered lots, then factory managers are forbidden to park anywhere:

$\forall fm: FactoryManager, \forall lot: ParkingLot\ (\mathbf{P}(park(fm, lot))$

$\Rightarrow \neg numbered\ (lot) \wedge numbered\ (lot)) \vdash$

$\forall fm: FactoryManager, \forall lot: ParkingLot(\mathbf{P}(park(fm, lot)) \Rightarrow \mathbf{false})$

$\vdash \forall fm, \forall lot: ParkingLot(\mathbf{F}\ (park(fm, lot)))$.

We fix this example after we discuss upward inheritance in the next section.

5.2. UPWARD INHERITANCE

To infer something about upward inheritance, we must add closure assumptions.

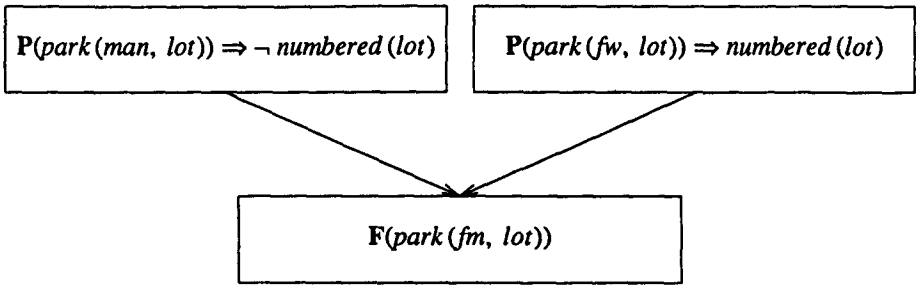

Fig. 7. Unintended constraint inheritance.

DEFINITION 5.7

1. If $type(x, \tau_1) \Leftrightarrow type(x, \tau_2) \vee type(x, \tau_3)$, then τ_1 is called the *cover* of τ_2 and τ_3, and we write $\tau_1 = \tau_2 \vee \tau_3$.
2. If $type(x, \tau_4) \Leftrightarrow type(x, \tau_2) \wedge type(x, \tau_3)$, then τ_4 is called the *intersection* of τ_2 and τ_3, and we write $\tau_4 = \tau_2 \wedge \tau_3$.

The following theorem is only stated for covers, but an obvious analogon holds for intersections. We write $\tau_2 \sqcup \tau_3$ for the least upper bound of τ_2 and τ_3 (see the appendix).

THEOREM 5.8

1. If $\tau_1 = \tau_2 \vee \tau_3$, then $\tau_1 = \tau_2 \sqcup \tau_3$.
2. $\tau_1 = \tau_2 \vee \tau_3$ iff $\| \tau_1 \|_w = \| \tau_2 \|_w \cup \| \tau_3 \|_w$ for all $w \in \mathscr{PW}$.
3. $\tau_2 \vee \tau_3 = \tau_3 \vee \tau_2$.

Proof

1. We have

$$\forall x(type(x, \tau_2)) \Rightarrow \forall x(type(x, \tau_2) \vee type(x, \tau_3)) \Rightarrow \forall x(type(x, \tau_1)),$$

so $\tau_2 \leqslant \tau_1$, and analogously $\tau_3 \leqslant \tau_1$. Furthermore, if there is a $\tau' \in \mathrm{T}$ with $\tau_2 \leqslant \tau'$ and $\tau_3 \leqslant \tau'$, then

$$\forall x(type(x, \tau_2) \vee type(x, \tau_3) \Rightarrow type(x, \tau')),$$

so

$$\forall x(type(x, \tau_1) \Rightarrow type(x, \tau')),$$

so $\tau_1 \leqslant \tau'$. So τ_1 is the least upper bound of τ_2 and τ_3.

2. By the truth definition for $\vee$ in $type(x, \tau_2) \vee type(x, \tau_3)$.
3. Trivial. □

Note that the converse of 1 is not true. If $Vehicle = Car \sqcup Airplane$, then we would have $Vehicle = Car \vee Airplane$ only if $\| Vehicle \|_w = \| Car \|_w \cup$

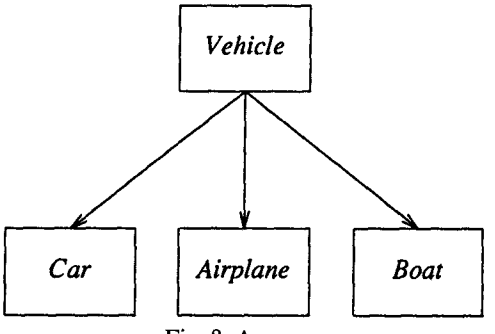

Fig. 8. A cover.

$\| Airplane \|_w$ in all possible worlds. But there may be *Boat*s that are *Vehicle*s. If all vehicles are cars, boats, or airplanes, then

$\| Vehicle \|_w = \| Car \|_w \cup \| Airplane \|_w \cup \| Boat \|_w$

for all $w \in \mathscr{P}\mathscr{W}$. We then have

$Vehicle = Car \sqcup Airplane \sqcup Boat = Car \sqcup Airplane = Airplane \sqcup Boat$
$= Car \sqcup Boat.$

See fig. 8, where the arrows indicate the direction of inheritance, from larger to smaller types.

Note secondly that we can omit the brackets in $Car \sqcup Airplane \sqcup Boat$, because $Car \sqcup Airplane$, $Airplane \sqcup Boat$ and $Car \sqcup Boat$ all exist. On a poset, $\sqcup$ (and $\sqcap$) is in general a partial operator, and we can write $\tau_1 \sqcup \tau_2 \sqcup \tau_3$ only if $(\tau_1 \sqcup \tau_2) \sqcup \tau_3$ and $\tau_1 \sqcup (\tau_2 \sqcup \tau_3)$ both exist and are equal. In general, if these exist, then they are equal (see appendix), and in the example, they happen to exist. On the other hand, we cannot write meaningfully

$Vehicle = Car \vee Airplane \vee Boat,$

because the appropriate covering types $Car \vee Airplane$, $Car \vee Boat$, and $Airplane \vee Boat$ do not exist in our example [2].

THEOREM 5.9

Let in figure 2, $\tau_1 = \tau_2 \vee \tau_3$. Then, if

$\forall x : \tau_2(\Phi_2(x) \Rightarrow \Psi(x))$ and $\forall x : \tau_3(\Phi_3(x) \Rightarrow \Psi(x))$

then

$\forall x : \tau_1(\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x)),$

and if

$\forall x : \tau_2(\Phi(x) \Rightarrow \Psi_2(x))$ and $\forall x : \tau_3(\Phi(x) \Rightarrow \Psi_3(x))$

then

$\forall x : \tau_1(\Phi(x) \Rightarrow \Psi_2(x) \vee \Psi_3(x)).$

Proof

By $\tau_1 = \tau_2 \vee \tau_3$, we can conclude

$\forall x(type(x, \tau_1) \Rightarrow type(x, \tau_2) \vee type(x, \tau_3)).$

Then we have

$\forall x : \tau_2(\Phi_2(x) \Rightarrow \Psi(x)) \wedge \forall x : \tau_3(\Phi_3(x) \Rightarrow \Psi(x))$
$\Leftrightarrow \forall x(type(x, \tau_2) \Rightarrow (\Phi_2(x) \Rightarrow \Psi(x))) \wedge \forall x(type(x, \tau_3) \Rightarrow (\Phi_3(x) \Rightarrow \Psi(x)))$

[2] We are careful to avoid the phrase "associative operator" in the above, for "associative" can be defined in several, non-equivalent ways for partial operators, and we don't want to commit ourselves to any of these ways.

$$\Leftrightarrow \forall x(type(x, \tau_2) \vee type(x, \tau_3) \Rightarrow (\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x)))$$

$$\Leftrightarrow \forall x(type(x, \tau_1) \Rightarrow (\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x))).$$

The second part is analogous. □

There is no corresponding result for single upward inheritance from τ_4 to τ_2 and τ_3, as shown by the following fact.

FACT 5.10

In fig. 2, There is no upward inheritance from τ_4 to τ_2 and τ_3, even if $\tau_4 = \tau_2 \wedge \tau_3$.

Proof

We have

$$\forall x : \tau_4(\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x))$$

$$\Leftrightarrow \forall x(type(x, \tau_4) \Rightarrow (\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x)))$$

$$\Leftrightarrow \forall x(type(x, \tau_2) \wedge type(x, \tau_3) \Rightarrow (\Phi_2(x) \wedge \Phi_3(x) \Rightarrow \Psi(x)))$$

$$\Rightarrow \forall x((type(x, \tau_2) \Rightarrow (\Phi_2(x) \Rightarrow \Psi(x))) \wedge (type(x, \tau_3) \Rightarrow (\Phi_3(x) \Rightarrow \Psi(x)))).$$

□

The reason for this asymmetry lies in the simple fact that inheritance is basically the implication relation, and this is an asymmetrical relation. The reason that we need a closure assumption for multiple upward inheritance but not for any case of downward inheritance is that inheritance goes downwards, not upwards.

We can summarize the theorem by the slogan

> If $\tau_1 = \tau_2 \vee \tau_3$, necessary conditions inherit upwards to τ_1 disjunctively, and sufficient conditions inherit upwards to τ_1 conjunctively.

Thus, necessary conditions get weaker as we generalize, and sufficient conditions get stronger. To illustrate this, consider fig. 7 again. The analyst probably intended to specify *sufficient* preconditions instead of necessary preconditions. Sufficient preconditions get weaker as we specialize and stronger as we generalize, but this does not imply a contradiction in the example, as fig. 9 shows. (We assume that all employees are factory workers or managers, using inclusive or.)

As an example of the upward inheritance of **objective modalities**, take the example given in [3] which we started with in the introduction. Let

GradStudent $\leqslant$ *Student*,

Undergrad $\leqslant$ *Student*,

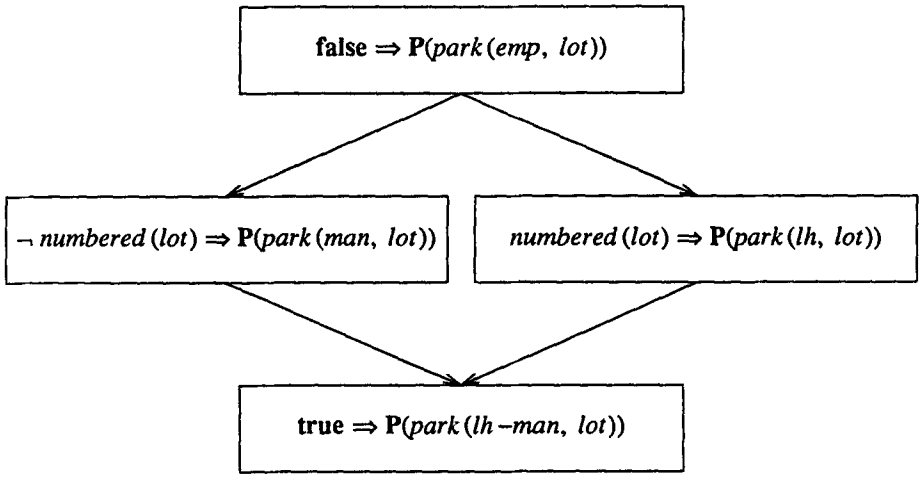

Fig. 9. Improvement of fig. 7.

$GradCourse \leqslant Course$, and

$UnderGradCourse \leqslant Course$, with

$Student = UnderGrad \vee GradStudent$,

$Course = UnderGradCourse \vee GradCourse$,

then the example is formalized by the following axioms.

$$\forall s: Student,\ c: Course\ (\mathbf{POS}(enroll(s, c)) \Rightarrow \neg full(c)) \tag{40}$$

$$\forall u: Undergrad,\ c: Course\ (\mathbf{POS}(enroll(u,c)) \Rightarrow (t < deadline) \wedge (enrollments(c) < undergrad\text{-}max)) \tag{41}$$

$$\forall u: Undergrad,\ gc: GradCourse\ (\mathbf{POS}(enroll(u, gc)) \Rightarrow permission(u, gc)) \tag{42}$$

$$\forall u: UnderGrad,\ uc: UnderGradCourse\ (\mathbf{POS}(enroll(u, uc)) \Rightarrow has\text{-}preparation(u, uc)) \tag{43}$$

$$\forall g: GradStudent,\ uc: UnderGradCourse,\ n: Natural\ (\mathbf{POS}(enroll(g, uc)) \Rightarrow year(g, n) \wedge n > 3). \tag{44}$$

We now know that the necessary enrollment condition $\neg full(c)$ is inherited conjunctively by all types of students. All preconditions in the example are necessary preconditions. Because they are inherited upwards disjunctively, we also know that their upward inheritance causes no inconsistencies or impossible preconditions for more general types.

6. Discussion

6.1. THE DISTINCTION BETWEEN NECESSARY AND SUFFICIENT PRECONDITIONS

The strange experience we had while writing this paper is that constraint inheritance is, or should be, easy, because it concerns only the logic of the implication sign, but that at the same time, constraint inheritance is an extremely slippery subject, because our common sense supplies so many assumptions that are unwarranted by formal logic. A number of unwarranted assumptions that we have encountered are:

1. When specifying preconditions, a temporal ordering is ascribed to the implication sign that is not there;
2. necessary and sufficient preconditions are confused;
3. an implicit, non-monotonic, completion assumption is often made.

These implicit assumptions supplied by common sense can be illustrated by the intuitive explanation of preconditions in Khosla et al. [10], who give a correct formal semantics of dynamic logic, but paraphrase the constraint

$$\neg \mathit{Lecturer}(\mathit{TOM}) \Rightarrow [\mathit{Hire}(\mathit{TOM})]\,\mathit{Lecturer}(\mathit{TOM})$$

informally as

"Tom can only be hired as a lecturer if he is not already one."

The "only if" phrase shows that what is really meant is

$$[\mathit{Hire}(\mathit{TOM})]\,\mathit{Lecturer}(\mathit{TOM}) \Rightarrow \neg \mathit{Lecturer}(\mathit{TOM}),$$

illustrating 2 and 3 above. Furthermore, they define a precondition as

"that assertion which must be true before the update can be applied",

which illustrates 1 above, if we assume that reading a temporal order into *imply* causes us to see the formula left of it as a precondition, and the formula right of it as a postcondition. Perhaps we should reiterate that only the box operator $[\alpha]$ implies a temporal ordering.

6.2. CONSTRAINT COMPLETION

A number of CM specification languages, such as TAXIS [3] and ACM/PCM [4], allow the specification of preconditions for actions. The logic of these preconditions is such that if they are not satisfied, the action is not performed. Thus, they implement the informal statement

1. $\neg \Psi \Rightarrow (\alpha$ is not executed).

This is not formalizable in $\mathit{OTL}_{\mathit{Deon}}$. Consider the following candidates.

2. $[\alpha]\Phi \Rightarrow \Psi$. "(If you *would* execute α, then afterwards, Φ necessarily holds) implies that currently, Ψ holds." This is, on the one hand, saying too much, because it mentions Φ, and on the other, saying too little, because it is a counterfactual statement, whereas 1 says that α actually occurs.

3. $\mathbf{POS}(\alpha) \Rightarrow \Psi$. "If α can be executed, then currently, Ψ holds." This eliminates Φ, but does not yet introduce the idea of actuality.
4. $\mathbf{NEC}(\alpha) \Rightarrow \Psi$. "If α will necessarily be executed, then currently, Ψ holds." This contrasts with 1, because there, α could possibly not occur, but here, we specify that α necessarily occurs only if Ψ holds.
5. $\mathbf{P}(\alpha) \Rightarrow \Psi$. "If α is permitted, then currently, Ψ holds." This comes as close as we can get in OTL_{Deon} to 1, for it is equivalent to $\neg\Psi \Rightarrow \mathbf{F}(\alpha)$. The difference is still that 1 carries a connotation of actuality that we have not yet captured. Instead, we have formalized only the statement that α is not permitted to occur if Ψ does not hold. What we would like is the following:
6. $\mathbf{EXEC}(\alpha) \Rightarrow \Psi$. "If α is to be executed, then currently Ψ holds." We simply note here that we have not given a formal semantics to the statement that, from the range of possible next actions, α will actually be executed. The opposition actuality/possibility is more informative than the opposition necessity/possibility and must await future formalization.

So far, we merely noted that we have not yet formalized the intention of some information analysts completely, when they specify preconditions of actions. Next, consider what we *have* formalized with a constraint like 5 above (or 6, if we would have a formal semantics for it). We have specified by 5 only when α is forbidden (when currently $\neg\Psi$ holds), but not when α it is permitted. However, it is implicitly assumed in languages like TAXIS and ACM/PCM that Ψ is, not only a necessary precondition, but the *strongest* necessary precondition (modulo logical equivalence) for α to be permitted. We have shown in section 4 that the strongest necessary precondition is actually the same as the weakest sufficient precondition, and that both are equivalent to the *weakest precondition of* α (possibly with respect to a postcondition). So the analyst apparently intends

$$\mathbf{P}(\alpha) \Leftrightarrow \Psi, \tag{44}$$

although he or she has specified

$$\mathbf{P}(\alpha) \Rightarrow \Psi \tag{45}$$

Apparently there is a kind of hidden "completion" of the preconditions, analogous to predicate completion in Prolog. We claim that this hidden assumption should be made explicit, so that the intention of the analyst agrees better with what he or she has specified. Moreover, if we do this, another type of hidden assumption, which says whether a type covers its subtypes, is made explicit as well, because constraint completion must be preceded by a process which we call *constraint collection*. We get the following two steps.

1. After having specified a set of constraints for a taxonomic network, all constraints that accrue to a type should be *collected.* This means that for each Ψ, the necessary and/or sufficient conditions Φ_τ specified for it for different τ's should be collected using upward and downward inheritance.
2. *Complete* the resulting conditions for Ψ by replacing the implication by an equivalence.

The aim of step 1 is to find out if what we have specified is consistent. Syntactically, the collecting process can be carried out algorithmically. The semantic problem of determining to which formulas a condition is equivalent, and of simplifying the resulting conditions, is in general undecidable. Step 1 may require the addition of *covering axioms* for some types in order to facilitate upward inheritance [3]. Different sets of covering axioms will yield different outputs. Assuming a particular set of covering axioms, the resulting constraint set is logically equivalent (using the axioms of dynamic logic) to the input constraint set. The result of step 1 is that we have a necessary condition and a sufficient condition for Ψ,

$\Psi \Rightarrow \Phi_{nec}$ and $\Phi_{suf} \Rightarrow \Psi$.

The aim of step 2 is to express the intention of the analyst more accurately. Doing this, we really commit the fallacy known in Aristotelian logic as the *fallacy of the consequent* ([9], p. 596), which consists of assuming that a condition and its consequent are convertible. There are often situations where one wants to make this fallacy, but when we do that, we should at least be aware of its problems. First of all, completion leads to

$\Phi_{suf} \Leftrightarrow \Psi \Leftrightarrow \Phi_{nec}$,

and this may simply be inconsistent. Secondly, even if it is consistent, it is in any case a non-monotonic operation, for the result implies the input but not vice versa. It is a generalization of predicate completion of Prolog [6,13] and reminiscent of McCarthy's [14] circumscription. In fact, predicate completion first collects the sufficient conditions that are given for a predicate $P(x)$, which gives

$E_1(x) \vee \cdots \vee E_n(x) \Rightarrow P(x)$,

and then minimizes the extension of $P(x)$ by taking the completion

$E_1(x) \vee \cdots \vee E_n(x) \Leftrightarrow P(x)$.

This is a special case of completion as described above. Constraint completion has the same problems as predicate completion. For example, we have $\Phi \Leftrightarrow (\Phi \vee \Phi) \Leftrightarrow (\neg\Phi \Rightarrow \Phi)$. So if Φ is any integrity constraint, then we can consistently add $\neg\Phi \Rightarrow \Phi$ to the specification, saying that $\neg\Phi$ is a sufficient condition for Φ. But then constraint completion gives us an inconsistent specification with

$\neg\Phi \Leftrightarrow \Phi$.

6.3. EXCEPTION SPECIFICATION

The archetypical case of exception specification is that of non-flying penguins. From

$Penguins \leqslant Bird$, (46)

[3] The addition of covering axioms agrees with an ancient Aristotelian prescription for how to design taxonomies, that *each subdivision should be exhaustive* ([9], p. 117) and ([20], p. 52).

$\forall b$: *Bird* (**POS**(*fly*(b))), (47)

$\forall p$: *Penguin* (**IMP**(*fly*(p))) (48)

we can derive

$\forall p$: *Penguin* (**IMP**(*fly*(p)) $\wedge$ **POS**(*fly*(p))), (49)

which implies $\forall p$: *Penguin* (**false**). This is inconsistent if there is at least one penguin.

The inconsistency is caused by the fact that we view penguins as an exception to a rule. We specify the rule (47) for all birds, whereas it is really a rule applicable only to a subtype of birds. If we specify it as such, the inconsistency disappears. For example, we can specify

Penguins $\leqslant$ *Bird*,

NonPenguins $\leqslant$ *Bird*,

$\forall n$: *NonPenguin* (**POS**(*fly*(n))),

$\forall p$: *Penguin* (**IMP**(*fly*(p))),

and the inconsistency has disappeared.

However, this is unsatisfactory because we had to add a rest-category to the taxonomy for which the problem may very well be repeated if our knowledge about this category increases. A more elegant solution has been suggested by McCarthy in a later paper [15]. To each rule, an *abnormality* predicate is added:

$\forall b$: *Bird* ($\neg$*AbnormalBird*(b) $\Rightarrow$ **POS**(*fly*(b))), (50)

$\forall p$: *Penguin* ($\neg$*AbnormalPenguin*(p) $\Rightarrow$ **IMP**(*fly*(p))), (51)

$\forall b$: *Penguin* (*AbnormalBird*(b)). (52)

We then apply circumscription to minimize the extension of the abnormality predicates. In this case it comes down to strengthening (52) to

$\forall p$(*type*(p, *Penguin*) $\Leftrightarrow$ *AbnormalBird*(p)). (53)

The advantage of this method is that we can easily extend the specification without having to retract or change rules specified earlier. (The axioms implied by the circumscription operation will of course change, but these are not explicitly specified by the designer). So if we add

Ostrich $\leqslant$ *Bird*, (54)

$\forall p$: *Ostrich* (*AbnormalBird*(p)), (55)

to (50)–(52), then circumscription gives us that

$\forall b$ (*type*(b, *Penguin*) $\vee$ *type*(b, *Ostrich*) $\Leftrightarrow$ *AbnormalBird*(b)) (56)

is valid, which is not equivalent to the result (53) of circumscription on the abnormality predicates in the original theory.

With the use of abnormality predicates, we can specify exceptions to the rules without having to change them in the CM specification. The price is that we need a non-monotonic operation to circumscribe the set of abnormal cases.

Instead of constraint completion as sketched above, we can apply a dynamic variant of circumscription to specifications in OTL_{Deon}. Very sketchily, one would proceed as follows. Using an abnormality predicate for each type/action pair specified in the CM specification, we get the following canonical form for sufficient preconditions (specified explicitly or derived non-monotonically):

$$\forall x : \tau(\Phi \wedge \neg EXC : \tau : \alpha(x) \Rightarrow \mathbf{P}(\alpha)),$$

where $EXC : \tau : \alpha$ stands for a predicate introduced by the analyst that should be read as "x is not an exceptional object of type τ with respect to action α." For example,

$$\forall x : \mathit{Student}\ (\mathit{graduate}(x) \wedge \neg EXC : \mathit{student} : \mathit{enroll}(x) \Rightarrow \mathbf{P}(\mathit{enroll}(x))).$$

If ϕ specifies an exception to this rule, we could add the axiom

$$\forall s : \mathit{Student}(\phi(x) \Rightarrow EXC : \mathit{student} : \mathit{enroll}\ (x) \wedge \mathbf{F}(\mathit{enroll}(x))).$$

6.4. METHODOLOGICAL CONSEQUENCES

Psychological research has shown that taxonomies are learnt neither top-down nor bottom-up, but from the middle out [21]. When a novice starts learning the structure of the UoD, he or she will usually start at the *basic level*, which is the level at which objects have the largest number of discriminating characteristic with respect to their neighbors in the taxonomy, or the level of the types of objects that are most frequently handled. Only later, finer distinctions and less frequently encountered categories of objects are added. Conversely, when UoD specialists ("domain specialists") explain the structure of the UoD to novices, they start with basic level objects. For example, when asked to mention a typical piece of furniture, subjects typically mention a chair, and not "an object to sit on" (which is more general than the concept of a chair, and includes couches as well), nor "a kitchen chair" (which is lower down the taxonomy).

Information analysts are usually novices with respect to the UoD of which they must specify a CM. This means that in general, we tend to specify the constraints for τ_2 and τ_3 in figure 1 first, and then proceed to add τ_1 and τ_4 to the taxonomy, and specify their constraints. The methodological consequences of the theorems about upward and downward inheritance are then that

1. they tell us if and how the constraints specified already must be "taken along" to newly added types, and
2. they make clear when we should require of a type that it covers its subtypes, viz. if we want a constraint to inherit upwards.
3. Furthermore, they allow us to deduce consequences from our specifying preconditions as necessary or sufficient preconditions, so that we can

express more clearly and accurately what we want to specify. For example, if we want a condition to be stronger for more specialized types, it must be a necessary condition, and if we want it to be weaker when we go down in the taxonomy, it should be a sufficient condition. But whichever choice we make, this has further consequences for the way constraints interact in multiple downward or upward inheritance.

4. Finally, we also showed that finding sufficient preconditions for actions is usually difficult, for we can never be sure to have found all possible cases. The methodological consequence of this is that we should avoid specifying sufficient preconditions as far as possible, and stick to necessary preconditions only. (This implies avoiding constraint completion as well.)

As an illustration of 1, we noted several times already that constraints should be specified as high in the taxonomy as possible (but not higher, witness the non-flying penguin example). This is an easy consequence of theorem 5.4, for if in fig. 2

$\tau_1 = \tau_2 \vee \tau_3$ and $\forall x: \tau_2\Phi(x)$ and $\forall x: \tau_3\Phi(x)$,

then we have

$\forall x: \tau_1\Phi(x)$.

The set of constraints specified for a cover should thus include the intersection of the sets of constraints specified for its subtypes.

7. Summary and conclusions

In section 1, we motivated the usefulness of deontic logic for constraint specification, in particular for the specification of IC's that may be violated by the UoD. In section 2, we briefly introduced the language L_{Deon}, a deontic extension of dynamic logic which can be used to specify static, dynamic, and deontic IC's. We then introduced in section 3 types to L_{Deon} as special constants, which gave us the language TL_{Deon}, and added an ordering on type names in section 5, yielding the language OTL_{Deon}. In this language, a taxonomy can be specified.

In section 4, we classified constraints, particularly dynamic constraints, with respect to whether they specify pre- or postconditions, and the kind of dynamic modality for which a precondition is specified. Inheritance was studied in section 5, where it is shown that necessary preconditions for actions inherit downwards conjunctively and, under a covering assumption, upwards disjunctively. Sufficient preconditions for actions inherit downwards disjunctively and, under a covering assumption, upwards conjunctively. These facts are illustrated with a large number of examples.

The main contributions of this paper are
- the distinction between roles and natural kinds in section 4, which allows us to specify preconditions for role changes explicitly;
- the distinction between necessary and sufficient (pre)conditions in section 4,
- an exposition of the role of type covering assumptions in upward inheritance.

The discussion in section 6 is more tentative and explores the connection with some AI approaches such as circumscription and the specification of exceptions, as well as some implications for conceptual modeling methodology. We should mention here that one of the topics of further research is making OTL_{Deon} executable.

In section 6.1 we noted that IC inheritance is at once elementary and difficult. More in particular, the *principle* of basic inheritance as stated in theorem 5.4 is a direct consequence of the logic of the implication sign, but the *practice* of constraint specification is fraught with unexpected problems, because IC's do not behave as our common sense thinks they should. This made one of us think (Wieringa) of the advice he got from his father-in-law when renovating his house: "Plumbing is easy, the only thing you have to know is, Water flows downwards." Similarly, constraint inheritance is easy, the only thing you have to know is, IC's inherit downwards. But somehow, like plumbing, practice is considerably more difficult than this principle.

Appendix

ELEMENTARY LATTICE THEORY

The following is based on chapters 1 and 5 of Birkhoff [2]. A set A with a partial order $\leqslant$ is denoted $(A, \leqslant)$. Partially ordered sets are also called posets.

DEFINITION A.1

Let $(A, \leqslant)$ be a poset.

1. An *upper bound* of a subset $X \subseteq A$ is an $a \in A$ with $x \leqslant a$ for all $x \in X$. The *supremum* (or join or lowest upper bound) of X in A, denoted $\sqcup(X)$, is an $a \in A$ smaller then every upper bound of X. Dual definitions can be given for *lower bounds* and the *infimum* (or meet or greatest lower bound) $\sqcap(X)$ of X.
2. $(A, \leqslant)$ is a *join semi-lattice* if any two elements $a_1, a_2 \in A$ have a supremum $a_1 \sqcup a_2 \in A$, and it is called a *meet semi-lattice* if any two elements $a_1, a_2 \in A$ have an infimum $a_1 \sqcup a_2 \in A$. $(A, \leqslant)$ is a *lattice* if it is a join- and meet semilattice.
3. $(A, \leqslant)$ is *complete* if every subset has a supremum and an infimum in A.

Remarks

1. The supremum of $X \subseteq A$ does not need to exist, even if A is finite. Take for example $A = \{a_1, a_2, a_3\}$ with $a_1 \leqslant a_2$. Then $\{a_2, a_3\}$ has no supremum.
2. If the supremum of $X \subseteq A$ exists, it is unique (this follows from the antisymmetry of $\leqslant$).
3. Because $A \subseteq A$, every non-empty complete lattice has a top element, denoted $\top$, and a bottom element, denoted $\bot$.
4. Every finite lattice is complete.

References

[1] J.W. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, 1980).

[2] G. Birkhoff, Lattice theory, Amer. Math. Soc. Colloquium Publ. 25 (1967).

[3] A. Borgida, J. Mylopoulos and H.K.T. Wong, Generalization-specialization as a basis for software specification, in: *On Conceptual Modelling*, Brodie et al. (eds.) (1984) pp. 87–114.

[4] M.L. Brodie and D. Ridjanovic, On the design and specification of database transactions, in *On Conceptual Modelling*, Brodie et al. (eds.) (1984) pp. 277–312.

[5] M.L. Brodie, J. Mylopoulos and J.W. Schmidt (eds.), *On Conceptual Modelling* (Springer, 1984).

[6] K.L. Clark, Negation as failure, in: *Logic and Databases*, H. Gallaire and J. Minker (eds.) (Plenum Press, 1978) pp. 293–322.

[7] F.P.M. Dignum, A language for modelling knowledge bases, Ph.D. Thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam (1989).

[8] D. Harel, Dynamic logic, in: *Handbook of Philosophical Logic*, vol. 2, D.M. Gabbay and F. Guenther (eds.) (Reidel, 1984).

[9] H.W.B. Joseph, *An Introduction to Logic*, 2nd rev. ed. (Oxford, 1916).

[10] S. Khosla, T.S.E. Maibaum and M. Sadler, Database specification, in: *Database Semantics (DS-1)*, T.B. Steel, Jr., and R. Meersman (eds.) (North-Holland, 1986) pp. 141–158.

[11] S. Kripke, Identity and necessity, in: *Naming, Necessity and Natural Kinds*, S.P. Schwartz (ed.) (Cornell University Press, 1977) pp. 66–101.

[12] S. Kripke, *Naming and Necessity*, 2nd ed. (Basil Blackwell, 1980).

[13] J.W. Lloyd, *Foundations of Logic Programming* (Springer, 1984).

[14] J. McCarthy, Circumscription – A form of non-monotonic reasoning, Art. Int. 13 (1980) 27–39.

[15] J. McCarthy, Applications of circumscription to formalizing common-sense knowledge, Art. Int. 28 (1986), 89–116.

[16] J.-J.Ch. Meyer, A simple solution to the "deepest" paradox in deontic logic, Logique et Analyse 30 (1987) 81–90.

[17] J.-J.Ch. Meyer, A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic, Notre Dame J. Formal Logic 19 (1988) 109–136.

[18] J.-J.Ch. Meyer, H. Weigand and R.J. Wieringa, A specification language for static, dynamic and deontic integrity constraints, *2nd Symp. on Mathematical Fundamentals of Database Systems*, J. Demetrovics and B. Thalheim (eds.), Springer Lecture Notes in Computer Science 364 (1989) pp. 347–366.

[19] H. Putnam, Is semantics possible?, in: *Language, Belief, and Metaphysics*, H.E. Kiefer and M.K. Munitz (eds.) (University of New York Press, 1970) pp. 50–63.

[20] N. Rescher, *Introduction to Logic* (St. Martin's Press, 1964).

[21] E. Rosch, Principles of categorization, in: *Cognition and Categorization*, E. Rosch and B.B. Lloyd (eds.) (Lawrence Erlbaum, 1978) pp. 27–48.

[22] J.F. Sowa, Knowledge representation in databases, expert systems and natural language, *Artificial Intelligence in Databases and Information Systems (DS-3)*, R.A. Meersman, Zongzhi Shi and Chen-Ho Kung (eds.) (North-Holland, 1990) pp. 17–43.

[23] R.J. Wieringa, J.-J. Ch. Meyer and H. Weigand, Specifying dynamic and deontic integrity constraints, Data and Krowledge Eng. 4 (1989) 157–189.

[24] R.J. Wieringa, Algebraic foundations for dynamic conceptual models, Ph.D. Thesis, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam (May 1990).