Deontic Concepts in the Algebraic Specification of Dynamic Systems: The Permission Case

Eva Coscia, Gianna Reggio

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Italy
Viale Dodecaneso, 35 – Genova 16146 – Italy
{ coseva,reggio } @ disi.unige.it

1 Introduction

The aim of our work is to analyse deontic logic to build up a formalism for the specification of dynamic systems. *Dynamic systems* are generic systems that are able to modify their own state during time, so processes and concurrent/reactive/ distributed systems are typical examples. Several researchers have recently pointed out how the applications of the deontic notions of *permission*, *prohibition* and *duty* or *obligation* are very useful in formal system modelling, see, e.g. [18] (for a survey and further references) and [11, 14, 15].

The most interesting feature of deontic logic is the clear separation between the *descriptive* and the *prescriptive* aspects of a specification: deontic formalisms allow us not only to describe how a system behaves, which is the external environment, which are the interactions with other components or users, but also to make a formal representation of which are the activities it is permitted or ought to do, which are the prohibitions that it ought not to violate. It is important to note how the prescriptive aspect of a specification gives us the possibility to make a distinction between the *ideal* and the *actual* (or *real*) behaviour of a system. Indeed, through deontic concepts we can impose norms describing what we expect from the system, what should be the optimal reactions in some cases and which activities ought to be (or ought not to be) executed: in one word, we make a picture of what is the ideal behaviour of the system. At the same time, since our norms just describe our concept of ideality, we leave open the possibility that a behaviour does not comply with the imposed norms, as in the real world something could go wrong and so real behaviours do not coincide with ideal ones. In a certain sense, even if we would like that the introduction of a token in a vending machine will be followed by the provision of a coffee, we know that such a request is unrealistic and just describes our concept of a perfect vending machine.

Another very useful concept related to norms is the concept of *violation*, see e.g. [14, 10]. When we are able to detect if a particular norm has been violated, we can specify error handling policy or introduce sanctions too.

We do not want to build a new formalism from scratch, but to introduce the deontic concepts into ADSs (Algebraic Dynamic Specifications, see, e.g. [1, 5, 9]), a framework to give algebraic (i.e. logic) specifications of dynamic systems at different levels of abstraction from the initial requirements till the complete design; ADSs are equipped with a specification methodology, software tools and have been applied to some relevant case studies, see e.g. [3, 20, 21].

In such applications we have frequently met the problem to specify: – that a system behaves differently in the normal cases and in the abnormal ones; – that a property holds except when some part breaks down; – what the system has to do when a user does not follow the instruction either by doing not allowed thing or by forgetting to do something (e.g. by pressing a button at the wrong time or by forgetting to refill). The last point is relevant for reactive system, which are usually intended to interact with users whose behaviour follows specific norms. These cases can be handled within ADSs, but result in very complex and long specifications, since normal and abnormal cases have to be considered together; for example, we may get a specification s.t. all formulae have the form either "if normal case then ..." or "if abnormal case then ..." and all liveness properties have the form "... eventually (... or part_1 breaks or ... or part_k breaks)".

An interesting point is that ADSs give a formal way to describe the dynamic aspect of system behaviour, by means of an arrow-predicate representing labelled transition capabilities of the system. Since we have already a complete and formal description of the evolution of a system, obligation, permission and prohibition can deal with *activities* as well as with *states* of the system, where the word activity may denote either simple interactions with the external world (labels) or transitions or sequences of transitions, called *abstract events* in [2].

A consequence of this assumption is that permission and obligation are completely independent, and unlike most deontic logics, where we have

$\neg P\phi \;\equiv\; O \neg \phi$,

with P and O deontic modalities for permission and obligation and ϕ formula, one cannot be derived from the other one. Indeed, if on the one hand the violation of the permission consists of the execution of a non-permitted transition and so we detect a violation every time *there is* a particular non-permitted activity, on the other hand, the violation of an obligation occurs when what has been requested *is not performed*, i.e. when in the behaviour of the system *there is not* a requested activity. Thus, the negation of the permission on the activity α is not equal to the obligation on $\neg\, \alpha$ (any activity different from α), since we are dealing with activities rather than with formulae. $\neg\, P\alpha$ means that someone is not allowed to do α, so it can do everything except α but it is allowed to stay idle too. On the other hand, $O\, \neg\, \alpha$ forces the system to move and to perform any activity except α, so it cannot stay idle.

For this reason we do not develop a framework dealing with both the deontic concepts, but treat them separately; this allows us to analyse each concept in more detail, without being bothered by the relations with the other one. We have thus developed one framework for permission (called P) and one for obligation (called O). For lack of room, in this paper we present only P; for a complete presentation of both P and O see [7, 6].

A relevant feature of P and O is that the classical paradoxes that have bothered a great deal of deontic logicians (see, e.g. [17, 16]) disappear. This is due mainly to the fact that activity and norms are separated and handled explicitly in the logics: we can express both that a system may perform some activity, and, apart, that an activity is permitted/obliged.

Both P and O formalisms are equipped with a deductive system, to be able to reason about specifications given using such logics. Furthermore such logics are sound formalisms, in the sense that they correspond to institutions, see [7].

It is worthwhile to note that these formalisms are rich and very powerful, as they supply operators to express different kinds of norms and, at the same time, to specify most of the requirements on dynamic systems. Moreover, in our opinion the use of deontic concepts makes it more easy and intuitive to formalize a system and to read a specification, since we are able to factorize normal and abnormal cases in a natural way.

To understand the real meaning of deontic logic and the usefulness of deontic concepts in formal system specification, we have analysed a great deal of work based on deontic concepts applied to computer science (see [10, 11, 14, 15, 18]). All these works gave us interesting hints for our formalisms.

In particular, our interpretation of deontic concepts is similar to the one in [10] in regard to the necessity for a clear distinction between *ought to do* and *ought to be* (i.e. between norms on state and norms on activities) and the use of violations associated with norms ([10, 14]); furthermore, as in [11], our aim is to associate a temporal dimension with norms.

In Sect. 2 we briefly summarize ADSs, in Sect. 3 we present the formalism P and in Sect. 4 we give the flavor of using a specification formalism based on P; some concluding remarks are in Sect. 5.

2 Algebraic Dynamic Specifications

Algebraic Dynamic Specifications, shortly ADSs, extend the algebraic specification of abstract data types, see [22], to the specification of types of dynamic systems. In this paper, with the term *dynamic system* we denote generic systems that are able to modify their own state during time, so processes and concurrent/reactive/distributed systems are typical examples.

2.1 The formal model for dynamic systems

In the ADS approach a dynamic system is modelled by a labelled transition system.

Definition 1. A *labelled transition system* (shortly *LTS*) is a triple $(S, L, \longrightarrow)$, where S and L are two sets whose elements are, respectively, the *states* and the *labels* of the system, while $\longrightarrow \subseteq S \times L \times S$ represents the *transition relation*. A triple (s, l, s') belonging to $\longrightarrow$ is called a *transition* and is usually written as $s \xrightarrow{l} s'$. □

A dynamic system D can be modelled by an LTS $(S, L, \longrightarrow)$ and an initial state $s_0 \in S$. The elements in S that can be reached starting from s_0 are the intermediate interesting states in the life of D, while the transition relation $\longrightarrow$ describes the capabilities of D to pass from one intermediate state to another one. So a transition $s \xrightarrow{l} s'$ has the following meaning: D in the state s *is able* to pass to the state s' by performing a transition whose interaction with the external world is described by the label l; thus l provides information both on the conditions on the external world making effective this capability and on the changes in the external world caused by this transition.

Concurrent dynamic systems, i.e. dynamic systems having cooperating components that are in turn other dynamic systems, can be modelled through particular LTSs obtained by composing other LTSs describing such components. In this paper, we do not put much emphasis on the fact that a system may have an internal concurrent structure, since we are interested in the analysis of its behaviour and the same considerations on deontic concepts can be applied even if the transitions of the systems are combinations of transitions of the components (for example a message exchange). See [5, 1] for handling concurrent systems by using ADSs.

2.2 Dynamic algebras

An LTS can be represented trough a many-sorted algebra with predicates (just a many-sorted 1^{st} order structure) A on a signature with two sorts, *state* and *label*, whose elements correspond to states and labels of the system, and a predicate $\longrightarrow : state \times label \times state$ representing the transition relation. In Appendix A we report the main definitions about algebras with predicates. The triple $(\mathrm{A}_{state}, \mathrm{A}_{label}, \longrightarrow^{A})$ is the corresponding LTS. Obviously we can have LTSs whose states are built by states of other LTSs (for modelling concurrent dynamic systems); in such a case we use algebras with different sorts corresponding to states and labels and with different predicates corresponding to transition relations.

In a formal model for dynamic systems we may need static elements too (for example, the data manipulated by the dynamic systems such as natural numbers); to handle these cases our algebras may have also sorts that just correspond to data and not to states or labels of LTSs.

The algebras corresponding to LTSs are called *dynamic algebras* and are formally defined as follows.

Definition 2.

- A *dynamic signature* $D\Sigma$ is a pair (Σ, DS), where:
  - $\Sigma = (S, OP, PR)$ is a predicate signature (see Appendix A);
  - $DS \subseteq S$ is the set of the *dynamic sorts*, i.e. sorts corresponding to dynamic systems (states of LTSs);
  - for all $ds \in DS$ there exist a sort $l\text{-}ds \in S - DS$ (the sort of the labels) and a predicate $\_ \xrightarrow{\_} \_ : ds \times l\text{-}ds \times ds \in PR$ (the transition predicate).
- A *dynamic algebra* on $D\Sigma$ is just a Σ-algebra with predicates; the term algebra $T_{D\Sigma}(X)$ is just $T_{\Sigma}(X)$, where X is a sort assignment on $D\Sigma$. $\square$

We give now some technical definitions on dynamic algebras that will be used in the following. Let DA be a $D\Sigma$-dynamic algebra and ds a dynamic sort of $D\Sigma$. $PATH(\mathrm{DA}, ds)$ denotes the set of the *paths* for the dynamic systems of sort ds, i.e. the set of all sequences of transitions having either of the two forms below:

(1) $\quad d_0\ l_0\ d_1\ l_1\ d_2\ l_2 \ldots d_n\ l_n\ \ldots$ (infinite path)

(2) $\quad d_0\ l_0\ d_1\ l_1\ d_2\ l_2 \ldots d_n\ l_n\ \ldots d_k \quad k \geq 0$ (finite path)

where for all $n \in \mathbb{N}$: $d_n \in \mathrm{DA}_{ds}$, $l_n \in \mathrm{DA}_{l\text{-}ds}$ and $(d_n, l_n, d_{n+1}) \in \longrightarrow^{\mathrm{DA}}$; moreover, in (2) for no d, l: $(d_k, l, d) \in \longrightarrow^{\mathrm{DA}}$ (there are no transitions starting from the final state of a finite path).

If σ is either (1) or (2) above, then

- $S(\sigma)$ denotes the first element of σ: d_0;
- $L(\sigma)$ denotes the second element of σ: l_0 (if it exists);
- $\sigma\lfloor n$ denotes the path $d_n\ l_n\ d_{n+1}\ l_{n+1}\ d_{n+2}\ l_{n+2}\ \ldots$ (if it exists).

2.3 A logic for ADSs

As for usual algebraic specifications an *ADS (Algebraic Dynamic Specification)* is a pair DSP $= (D\Sigma, AX)$ where $D\Sigma$ is a dynamic signature and AX is a set of formulae on $D\Sigma$.

In this case the axioms in AX must express both static properties of the data and of the dynamic systems, for which 1^{st} order logic is adequate, and properties on the activity of the dynamic systems, such as liveness or safety requirements; for this aim 1^{st} order logic is enriched with the combinators of the branching-time temporal logic with edge formulae, see [8, 9]. Below we present the resulting temporal logic for ADSs, called T.

Definition 3.
The family of the sets of the *path formulae*, denoted by $\{PF_{\mathsf{T}}(D\Sigma, X)_{ds}\}_{ds \in DS}$, and of the *formulae of* T, denoted by $F_{\mathsf{T}}(D\Sigma, X)$, on $D\Sigma = ((S, OP, PR), DS)$ and a sort assignment X are defined by multiple induction as follows. For each $s \in S$ and $ds \in DS$:

path formulae

- $[\,\lambda\, x \,.\, \phi\,] \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$ $\qquad x \in X_{ds},\ \phi \in F_{\mathsf{T}}(D\Sigma, X)$
- $\langle \lambda x \,.\, \phi\rangle \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$ $\qquad x \in X_{l\text{-}ds},\ \phi \in F_{\mathsf{T}}(D\Sigma, X)$
- $\pi_1\ \mathcal{U}\ \pi_2 \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$ $\qquad \pi_1, \pi_2 \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$
- $\neg\, \pi_1, \pi_1 \supset \pi_2, \forall\, x \,.\, \pi_1 \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$ $\qquad \pi_1, \pi_2 \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}, x \in X_s$

formulae

- $Pr(t_1, \ldots, t_n) \in F_{\mathsf{T}}(D\Sigma, X)$ $\qquad Pr\colon s_1 \times \ldots \times s_n \in PR,\ t_i \in T_{D\Sigma}(X)_{s_i}, i = 1 \ldots n$
- $t_1 = t_2 \in F_{\mathsf{T}}(D\Sigma, X)$ $\qquad t_1, t_2 \in T_{D\Sigma}(X)_s$
- $\neg\, \phi_1, \phi_1 \supset \phi_2, \forall\, x \,.\, \phi_1 \in F_{\mathsf{T}}(D\Sigma, X)$ $\qquad \phi_1, \phi_2 \in F_{\mathsf{T}}(D\Sigma, X), x \in X$
- $\triangle(t, \pi) \in F_{\mathsf{T}}(D\Sigma, X)$ $\qquad t \in T_{D\Sigma}(X)_{ds}, \pi \in PF_{\mathsf{T}}(D\Sigma, X)_{ds}$ $\square$

Definition 4. Let DA be a $D\Sigma$-algebra and V an evaluation of the variables in X in DA, then we define by multiple induction:

- the validity of a path formula π on a path $\sigma \in PATH(\mathrm{DA}, ds)$ in DA w.r.t. V (written $\mathrm{DA}, V, \sigma \models \pi$),
- the validity of a formula ϕ in DA w.r.t. V (written $\mathrm{DA}, V \models \phi$),

as follows ($t^{\mathrm{DA},V}$ denotes the interpretation of term t w.r.t. DA and V):

path formulae

- $\mathrm{DA}, V, \sigma \models [\,\lambda\, x \,.\, \phi\,]$ iff $\mathrm{DA}, V[S(\sigma)/x] \models \phi$
- $\mathrm{DA}, V, \sigma \models \langle \lambda x \,.\, \phi\rangle$ iff either $\mathrm{DA}, V[L(\sigma)/x] \models \phi$ or $L(\sigma)$ is not defined.
- $\mathrm{DA}, V, \sigma \models \pi_1\ \mathcal{U}\ \pi_2$ iff there exists $j > 0$ s.t. $\sigma\lfloor j$ is defined, $\mathrm{DA}, V, \sigma\lfloor j \models \pi_2$ and for each i s.t. $0 < i < j$, $\mathrm{DA}, V, \sigma\lfloor i \models \pi_1$

- $\mathrm{DA}, V, \sigma \models \neg \pi$ iff $\mathrm{DA}, V, \sigma \not\models \pi$
- $\mathrm{DA}, V, \sigma \models \pi_1 \supset \pi_2$ iff either $\mathrm{DA}, V, \sigma \not\models \pi_1$ or $\mathrm{DA}, V, \sigma \models \pi_2$
- $\mathrm{DA}, V, \sigma \models \forall\, x \,.\, \pi_1$ iff for each $v \in \mathrm{DA}_{ds}$, $\mathrm{DA}, V[v/x], \sigma \models \pi_1$

formulae

- $\mathrm{DA}, V \models Pr(t_1, \ldots, t_n)$ iff $(t_1^{\mathrm{DA},V}, \ldots, t_n^{\mathrm{DA},V}) \in Pr^{\mathrm{DA}}$
- $\mathrm{DA}, V \models t_1 = t_2$ iff $t_1^{\mathrm{DA},V} = t_2^{\mathrm{DA},V}$
- $\mathrm{DA}, V \models \neg \phi$ iff $\mathrm{DA}, V \not\models \phi$
- $\mathrm{DA}, V \models \phi_1 \supset \phi_2$ iff either $\mathrm{DA}, V \not\models \phi_1$ or $\mathrm{DA}, V \models \phi_2$
- $\mathrm{DA}, V \models \forall\, x \,.\, \phi$ iff for each $v \in \mathrm{DA}_s$, with s sort of x, $\mathrm{DA}, V[v/x] \models \phi$
- $\mathrm{DA}, V \models \triangle(t, \pi)$ iff
  for each $\sigma \in \mathit{PATH}(\mathrm{DA}, ds)$, with ds sort of t, s.t. $S(\sigma) = t^{\mathrm{DA},V}$, $\mathrm{DA}, V, \sigma \models \pi$

ϕ is *valid* in DA (written $\mathrm{DA} \models \phi$) iff $\mathrm{DA}, V \models \phi$ for all evaluations V. $\Box$

The formulae of T include the usual (hence static) ones of many-sorted 1^{st} order logic with equality; if $D\Sigma$ contains dynamic sorts, they include also formulae built with the transition predicates.
$\triangle(t, \pi)$ can be read as "for every path σ starting from the state denoted by t, the path formula π holds on σ". We do not model a single system but, in general, a type of systems, so there is not a single initial state but several of them, hence the need for an explicit reference to states (through terms) in the formulae built with $\triangle$.
$[\lambda\, x \,.\, \phi]$ holds on a path σ whenever ϕ holds at the first state of σ; similarly $\langle \lambda x \,.\, \phi \rangle$ holds on σ if either σ consists of a single state or ϕ holds at the first label of σ.

Finally, $\mathcal{U}$ is the so called strong future until operator.

In the above definitions we have used a minimal set of combinators; in practice, however, it is convenient to use other, derived, combinators; we list below those that we shall use in this paper.

- **true** , **false** , $\vee$, $\wedge$, $\exists$, $\equiv$, defined in the usual way
- $\bigtriangledown(t, \pi) =_{\text{def}} \neg \triangle (t, \neg \pi)$
  $\mathrm{DA}, V \models \bigtriangledown(t, \pi)$ iff there exists $\sigma \in \mathit{PATH}(\mathrm{DA}, ds)$, with ds sort of t, s.t. $S(\sigma) = t^{\mathrm{DA},V}$ and $\mathrm{DA}, V, \sigma \models \pi$
- $\Diamond\, \pi =_{\text{def}}$ **true** $\mathcal{U}\, \pi$ (eventually π)
  $\mathrm{DA}, V, \sigma \models \Diamond\, \pi$ iff there exists $i > 0$ s.t. $\sigma\lfloor i$ is defined and $\mathrm{DA}, V, \sigma\lfloor i \models \pi$
- $\Box\, \pi =_{\text{def}} \neg \Diamond \neg \pi$ (always π)
  $\mathrm{DA}, V, \sigma \models \Box\, \pi$ iff $\mathrm{DA}, \sigma\lfloor i, V \models \pi$ for all $i > 0$ s.t. $\sigma\lfloor i$ is defined
- $\bigcirc\, \pi =_{\text{def}} \langle \lambda x \,.\, \textbf{false} \rangle \vee (\textbf{false}\ \mathcal{U}\, \pi)$ (next π)
  $\mathrm{DA}, V, \sigma \models \bigcirc\, \pi$ iff either $\sigma\lfloor 1$ is not defined or $\mathrm{DA}, V, \sigma\lfloor 1 \models \pi$.
  Notice that $\langle \lambda x \,.\, \textbf{false} \rangle$ can be satisfied only by a path which consists of a single state, the initial one.

Each time in ϕ there are no free variables of dynamic sort except x, $[\lambda\, x \,.\, \phi]$ is abbreviated to $[\phi]$ and $[s = t]$ to $[t]$; analogously $\langle \lambda x \,.\, \phi \rangle$ and $\langle l = t \rangle$ are abbreviated to $\langle \phi \rangle$ and $\langle t \rangle$.

The *models* of DSP, $\mathit{Mod}(\mathrm{DSP})$, are the $D\Sigma$-algebras DA s.t. the axioms in AX are valid in DA.

We consider two different kinds of ADSs: *requirement* (for the starting and intermediate requirements of a system) and *design* (for the specification of the final design of a system), characterized by different semantics:

- the semantics of a requirement specification is the class of its models (loose semantics);
- the semantics of a design specification is the initial element in the class of its models, if any (recall that initial element is unique up to isomorphism).

A dynamic specification may not have an initial model, since it might contain axioms like: $t_1 = t_2 \ \vee\ t_1 = t_3$ or $\exists\, x\, .\, Pr(x)$; so we have to restrict the form of the axioms used in design specifications, by considering only *conditional* axioms having the following form: $\alpha_1 \ \wedge\ \ldots\ \wedge\ \alpha_n \ \supset\ \alpha$, where α_i and α are atoms i.e. either $Pr(t_1, \ldots, t_n)$ or $t = t'$.

Proposition 5. *Given an ADS* DSP $= (D\Sigma, AX)$ *whose axioms are conditional, then there exists (unique up to isomorphism)* $\mathrm{I}_{\mathrm{DSP}}$ *initial in* $Mod(\mathrm{DSP})$ *characterized by*

- *for all* $t_1, t_2 \in \mathrm{T}_{D\Sigma}$ *of the same sort* $\quad \mathrm{I}_{\mathrm{DSP}} \models t_1 = t_2$ *iff* $AX \vdash t_1 = t_2$*;*
- *for all* $Pr \in PR$ *and all* $t_1, \ldots, t_n \in \mathrm{T}_{D\Sigma}$ *of appropriate sorts* $\mathrm{I}_{\mathrm{DSP}} \models Pr(t_1, \ldots, t_n)$ *iff* $AX \vdash Pr(t_1, \ldots, t_n)$*;*

where $\vdash$ *denotes provability in the Birkhoff sound and complete deductive system for conditional axioms, whose rules are:*

$$t = t \qquad \frac{t = t'}{t' = t} \qquad \frac{t = t' \quad t' = t''}{t = t''}$$

$$\frac{t_i = t'_i \quad i = 1, \ldots, n}{Op(t_1, \ldots, t_n) = Op(t'_1, \ldots, t'_n)} \qquad \frac{t_i = t'_i \quad i = 1, \ldots, n \quad Pr(t_1, \ldots, t_n)}{Pr(t'_1, \ldots, t'_n)}$$

$$\frac{\alpha_1 \ \wedge\ \ldots\ \wedge\ \alpha_n \ \supset\ \alpha \qquad \alpha_i \quad i = 1, \ldots, n}{\alpha}$$

$$\frac{F}{\overline{V}(F)} \quad V\colon X \to \mathrm{T}_{\Sigma}(X)$$

$\overline{V}(F)$ *is* F *where each occurrence of a variable, say* x*, has been replace by* $V(x)$*.* □

A notion of "correct implementation" between ADSs has been given (see [5]) as follows: DSP is *implemented* by DSP$'$ with respect to α, a function from specifications into specifications, iff $Mod(\alpha(\mathrm{DSP}')) \subseteq Mod(\mathrm{DSP})$. The function α describes how the parts of DSP are realized in DSP$'$ (implementation as realization); while implementation as refinement is obtained by requiring inclusion of the classes of models. Notice that this definition applies whatever is the semantics for DSP$'$, if it is initial, then it has just one up to isomorphism model.

3 The Formalism P

3.1 Motivations

We want to extend ADSs with the possibility to establish exactly what are the activities of a system that, according to particular criteria not further specified, may be considered permitted (or acceptable). We can, therefore, "divide" the transitions of the system into permitted and non-permitted; where permitted/non-permitted may be intended: either right/wrong, or normal/abnormal, or usual/exceptional.

The most immediate solution is to factorize the transitions of the system through two predicates $\longrightarrow_P$ and $\longrightarrow_{NP}$ that, as the intuition suggests, characterize respectively the permitted and non-permitted transitions.

This distinction, obviously, leaves unchanged the capabilities of the system defined by means of $\longrightarrow$, since the predicate $\longrightarrow$ is the union of $\longrightarrow_P$ and $\longrightarrow_{NP}$, i.e. each transition of the system has to be qualified either as permitted or non-permitted. In a similar way, the intersection of $\longrightarrow_P$ and $\longrightarrow_{NP}$ must be the empty set, thus we exclude any conflict between permission and prohibition, as it is not meaningful to qualify a transition as permitted and prohibited at the same time.

This choice does not force in any way the system to perform only the permitted transitions. If on the one hand, the execution of permitted transitions guarantees the conformity of a behaviour to the ideal representation of the system we have, on the other hand we have also the possibility to completely define the capabilities of evolution that are less agreeable but nevertheless real. In a real system some activities may happen even if they are not desired by the designer, for example activities caused by bad uses, technological limits (and, thus, due to possible bad working, failures, breakdowns) or by the external world from which non-permitted intrusions could come.

From the distinction of transitions made via $\longrightarrow_P$ and $\longrightarrow_{NP}$, we can derive several definitions and concepts, listed below, on the paths (behaviours) of a system, which are useful to express relevant properties.

Safe paths, first introduced in [11], are the ones composed only of permitted transitions, so that they are considered *ideal* w.r.t. our classification of the system activity. With the term *unsafe*, we generically mark paths containing at least one non-permitted transition. This qualification seems to be too loose, as we do not make a distinction based on the kind and the quantity of non-permitted transitions made within unsafe paths; so it is also useful to characterize the *recovered* paths, i.e. the ones in which any non-permitted transition is eventually followed by a permitted one. We are thus guaranteed that the system does not continue to pass from an unsafe state to another one, behaving in an uncontrollable way; indeed after each sequence of non-permitted transitions, a permitted transition is performed, which restores an acceptable situation for the system.

Since we have to take into account that several non-permitted transitions might occur consecutively, it is too restrictive to require that the action of recovery occurs immediately, while it seems to be more realistic to require that it will occur in the future, without any temporal constraints.

If we limit ourselves to define transition capabilities of the system and to qualify them by means of $\longrightarrow_P$ and $\longrightarrow_{NP}$, again we are not able to express "behavioural" properties of the system, that is properties of liveness, fairness, ... about the whole behaviour of the system. Since these properties cannot be formulated using the 1st order logic, it is necessary to use a logic that allows us to tie the validity of particular formulae to paths of the system. In our particular case we have chosen to use the logic T presented in Sect. 2.3.

From the combination of temporal concepts and the deontic qualification we derive some new combinators, presented below.

$\underset{\text{Safe}}{\triangle}$ (in all safe paths) allows us to construct formulae requiring conditions only on safe paths of the system. The operator $\underset{\text{Safe}}{\nabla}$ (there exists a safe path), used in the following example, has a similar meaning. The ideal system simulating poker that allows the users to win money lacking breakdowns, failures or bugs, behaves in an "honest" way with respect to each user (i.e. it does not swindle tokens and pays for each winnings). For this system we can require that among all the safe paths, there exists *at least one* in which the user gets the biggest prize, corresponding to poker: $\underset{\text{Safe}}{\nabla}(st, \Diamond \langle Poker \rangle)$

In a similar way, $\underset{\text{Rec}}{\triangle}$ (in all recovered paths) allows us to express properties that must be satisfied by paths in which failures had been recovered.

While $\underset{\text{Safe}}{\triangle}$ allows us to completely describe the ideal system, by using $\underset{\text{Rec}}{\triangle}$ we can define in a complete way what we expect from a real system, i.e. of a system that, even if subject to failures, is able to remedy them and to behave correctly again.

So far, we have used the qualification safe and recovered to introduce particular properties on particular classes of paths. These properties are expressed by temporal path operators that allow us to formalize expressions like *eventually* or *next* and so on. Now it may be useful to enrich these expressions by imposing the condition that the properties are satisfied without infringing the permission structure defined on transitions. We may require that a property associated with a state is always satisfied by performing *only permitted transitions* or eventually but *in a permitted way* and so on. This kind of properties is expressed by means of $\mathcal{U}_{\mathcal{P}}$, $\Box_P$, $\Diamond_P$, $\bigcirc_P$ associating with the primitive temporal combinators the concept of permission.

3.2 The logic P

Definition 6. A P-*signature* $\mathsf{P}\Sigma$ is a dynamic signature (Σ, DS), where $\Sigma = (S, OP, PR)$, $DS \subseteq S$, s.t. for each $ds \in DS$, there exist two predicates: $\longrightarrow_P$, $\longrightarrow_{NP} : ds \times l\text{-}ds \times ds \in PR$. $\Box$

The models of P are particular dynamic algebras, in which each transition is qualified either as permitted or non-permitted and no transition is, at the same time, permitted and non-permitted.

Definition 7. A $\mathsf{P}\Sigma$-algebra PA is a $\mathsf{P}\Sigma$-dynamic algebra s.t. for all $ds \in DS$: $\longrightarrow^{\mathrm{PA}} = \longrightarrow_P{}^{\mathrm{PA}} \cup \longrightarrow_{NP}{}^{\mathrm{PA}}$ and $\longrightarrow_P{}^{\mathrm{PA}} \cap \longrightarrow_{NP}{}^{\mathrm{PA}} = \emptyset$. $\Box$

Since a P-signature $\mathsf{P}\Sigma$ and a $\mathsf{P}\Sigma$-algebra are, respectively, a dynamic signature and a dynamic algebra, the set of the P-*formulae*, denoted by $F_{\mathsf{P}}(\mathsf{P}\Sigma, X)$, is defined as in Def. 3, and the *validity of a formula ϕ in* PA *w.r.t.* V, written $\mathrm{PA}, V \models_{\mathsf{P}} \phi$, is defined as in Def. 4.

Above we have just given the basic logic; but the interesting formulae of P are combinations of temporal combinators with the predicates denoting the permitted and non-permitted transitions. Below we report the most relevant and frequently used, together with their semantics (formal or informal according to which is clearer).

- $\langle \lambda y \,.\, \phi \rangle_P \;=_{\rm def}$
  $\forall\, d, d', l \,.\, (([\lambda x \,.\, x = d] \wedge \langle \lambda y \,.\, y = l \rangle \wedge \bigcirc\, [\lambda x \,.\, x = d']) \supset (d \xrightarrow{l}_P d' \wedge \phi[l/y]))$
  $\mathrm{PA}, V, \sigma \models_{\mathsf{P}} \langle \lambda y \,.\, \phi \rangle_P$ iff either $(\mathrm{PA}, V[L(\sigma)/y] \models \phi$ and $(S(\sigma), L(\sigma), S(\sigma\lfloor 1)) \in\; \longrightarrow_P{}^{\mathrm{PA}})$ or $L(\sigma)$ is not defined.
- $\pi_1\, \mathcal{U}_{\mathcal{P}}\; \pi_2 \;=_{\rm def} \langle\, \mathbf{true}\, \rangle_P \;\wedge\; (\langle\, \mathbf{true}\, \rangle_P \;\wedge\; \pi_1)\; \mathcal{U}\; \pi_2$
  this formula holds on a path whenever the path is (composed by permitted transitions and satisfies π_1) until π_2 becomes true.
- $\Box_P\; \pi \;=_{\rm def} \langle\, \mathbf{true}\, \rangle_P \;\wedge\; \Box\; (\pi \;\wedge\; \langle\, \mathbf{true}\, \rangle_P)$
  this formula holds on a safe path always satisfying π.
- $\Diamond_P\; \pi \;=_{\rm def} \langle\, \mathbf{true}\, \rangle\, \mathcal{U}_{\mathcal{P}}\; \pi$
  this formula holds on a path whenever it has a point for which π holds and it is composed of permitted transitions until such a point.
- $\bigcirc_P\; \pi \;=_{\rm def} [\;\mathbf{false}\;]\, \mathcal{U}_{\mathcal{P}}\; \pi$
  this formula holds on a path whenever it starts with a permitted transition and π holds at the next point.
- $\mathbf{safe} \;=_{\rm def} \langle\, \mathbf{true}\, \rangle_P \;\wedge\; \Box\, \langle\, \mathbf{true}\, \rangle_P$
  this formula holds on a *safe path*, i.e. on a path consisting only of permitted transitions.
- $\mathbf{rec} \;=_{\rm def} (\Box\, \Diamond\, (\langle\, \mathbf{true}\, \rangle_P \;\wedge\; \neg\, \mathbf{stop}) \;\vee\; (\Diamond\, \bigcirc_P\; \mathbf{stop}) \;\vee\; (\bigcirc_P\; \mathbf{stop})$,
  where $\mathbf{stop} \;=_{\rm def} [\; \lambda\, x \,.\, \neg\, \exists\, y, z \,.\, x \xrightarrow{y} z\;]$
  this formula holds on a *recovered path*, i.e. on a path s.t. either at each point there is always a subsequent permitted transition or, eventually there is a permitted transition reaching a state from which the system cannot make more moves.

Obviously we may define in the same way all the temporal combinators associated with non-permitted transitions; for example, $\bigcirc_{NP}\; \pi$ holds on a path whenever it starts with a non-permitted transition and π holds at the next point.

- $\underset{\rm Safe}{\triangle}(t, \pi) \;=_{\rm def} \triangle(t, \mathbf{safe} \,\supset\, \pi) \qquad \underset{\rm Rec}{\triangle}(t, \pi) \;=_{\rm def} \triangle(t, \mathbf{rec} \,\supset\, \pi)$
  these formulae hold if the path formula π holds, respectively, on all safe and on all recovered paths starting with the state represented by the term t.

Now let us point out the difference between a formula like $\underset{\rm Safe}{\triangle}(t, \Diamond\; \pi)$ and $\triangle(t, \Diamond_P\; \pi)$. For example, consider how to formalize that a vending machine must give a drink after it receives a token in a correct way:

1) $\quad vm \xrightarrow{TOKEN}_P vm' \;\supset\; \underset{\rm Safe}{\triangle}(vm', \Diamond\, \langle SUPPLY(d) \rangle)$

2) $\quad vm \xrightarrow{TOKEN}_P vm' \;\supset\; \triangle(vm', \Diamond_P\, \langle SUPPLY(d) \rangle)$

While in 1) the use of $\underset{\rm Safe}{\triangle}$ helps us to describe completely the ideal behaviour of a vending machine (we impose that vm' eventually will give a drink in each safe path), 2) requires that vm' has only paths that are safe at least until it safely gives a drink and that surely it will give the drink.

3.3 A deductive system for P

First we show how any deductive system that is sound (complete) for T, see Sect. 2.3, may be extended to a system that is sound (complete) for P too.

Recall that a P-signature $\mathsf{P}\Sigma$ is just a particular dynamic signature, that the P formulae over $\mathsf{P}\Sigma$ are just the T formulae over $\mathsf{P}\Sigma$ and that the P models are a strict subclass of the $\mathsf{P}\Sigma$-dynamic algebras. Precisely, given a P-specification DSP = $(\mathsf{P}\Sigma, AX)$, then it is easy to prove that

$Mod_{\mathsf{P}}(\mathrm{DSP}) = Mod((\mathsf{P}\Sigma, AX \cup \{(1),(2)\}))$

where: $Mod(\mathrm{DSP})$ is the set of the models of DSP (i.e. all $\mathsf{P}\Sigma$-dynamic algebras satisfying AX) and $Mod_{\mathsf{P}}(\mathrm{DSP})$ is the set of P-algebras satisfying AX;

(1) $d \xrightarrow{l} d' \equiv d \xrightarrow{l}_P d' \vee d \xrightarrow{l}_{NP} d'$;

(2) $d \xrightarrow{l}_P d' \supset \neg\, d \xrightarrow{l}_{NP} d'$.

Thus any sound deductive system for T can be extended to a sound one for P just by adding (1) and (2), and the incompleteness result for T in [9] can be easily extended to P, so that P does not admit a complete effective deductive system.

The sound system for T presented in [9] and extended with the axioms (1) and (2) could be taken as the basic system for reasoning over P specifications. Below we present some formulae expressing sample properties of the deontic combinators of P which can be proved using the above deductive system.

$$\triangle(t, \langle\lambda x\,.\,\phi\rangle_P) \equiv \forall\, d', l\,.\, t \xrightarrow{l} d' \supset (t \xrightarrow{l}_P d' \wedge \phi[l/x])$$

$$\triangle(t, \mathbf{safe}) \equiv \forall\, d', l\,.\, t \xrightarrow{l} d' \supset (t \xrightarrow{l}_P d' \wedge \triangle(d', \mathbf{safe}))$$

$$\triangle(x, \mathbf{safe}) \supset \triangle(x, \mathbf{rec})$$

$$\underset{\text{Safe}}{\triangle}(x,\pi) \supset \underset{\text{Rec}}{\triangle}(x,\pi) \qquad \triangle(x,\pi) \supset \underset{\text{Safe}}{\triangle}(x,\pi) \wedge \underset{\text{Rec}}{\triangle}(x,\pi)$$

$$\triangle(x, \langle\lambda x\,.\,\phi\rangle_P) \supset \triangle(x, \langle\lambda x\,.\,\phi\rangle) \qquad \triangle(x, \pi_1\, \mathcal{U}_{\mathcal{P}}\, \pi_2) \supset \triangle(x, \pi_1\, \mathcal{U}\, \pi_2)$$

$$\triangle(x, \Box_P\, \pi) \supset \triangle(x, \Box\, \pi) \qquad \triangle(x, \Diamond_P\, \pi) \supset \triangle(x, \Diamond\, \pi)$$

$$\triangle(x, \bigcirc_P\, \pi) \supset \triangle(x, \bigcirc\, \pi)$$

$$\underset{\text{Safe}}{\triangle}(x, \langle\lambda x\,.\,\phi\rangle_P) \equiv \underset{\text{Safe}}{\triangle}(x, \langle\lambda x\,.\,\phi\rangle) \qquad \underset{\text{Safe}}{\triangle}(x, \pi_1\, \mathcal{U}_{\mathcal{P}}\, \pi_2) \equiv \underset{\text{Safe}}{\triangle}(x, \pi_1\, \mathcal{U}\, \pi_2)$$

$$\underset{\text{Safe}}{\triangle}(x, \Box_P\, \pi) \equiv \underset{\text{Safe}}{\triangle}(x, \Box\, \pi) \qquad \underset{\text{Safe}}{\triangle}(x, \Diamond_P\, \pi) \equiv \underset{\text{Safe}}{\triangle}(x, \Diamond\, \pi)$$

$$\underset{\text{Safe}}{\triangle}(x, \bigcirc_P\, \pi) \equiv \underset{\text{Safe}}{\triangle}(x, \bigcirc\, \pi)$$

3.4 Existence of the Initial Model for P-design Specifications

As we have seen in Prop. 5, a dynamic specification admits an initial model only if we restrict the form of the axioms: they should be conditional. However, these restrictions are no longer enough to guarantee that such an initial model is a P model: an axiom like $t \xrightarrow{l} t'$ admits two different non-isomorphic models, since in a model either $t \xrightarrow{l}_P t'$ holds or $t \xrightarrow{l}_{NP} t'$ holds but not both.

So another restriction becomes necessary: we consider only conditional formulae which do not contain the predicate $\longrightarrow$, but clearly may contain $\longrightarrow_P$ and $\longrightarrow_{NP}$. Thus when one specify the design of a system he must just define the permitted and the non-permitted activities.

Finally, note that, while conditional specifications on a dynamic signature cannot be inconsistent, P conditional specifications satisfying the above restrictions may be so, since it is possible to specify that a transition is both permitted and non-permitted (and that cannot happen in a P model).

Proposition 8. *Given a* P *conditional specification* $\mathrm{DSP} = (\mathsf{P}\Sigma, AX)$. *If*

- *the formulae in* AX *do not contain the* $\longrightarrow$ *predicates;*
- *for each* $ds \in DS$, $t, t' \in T_{\mathsf{P}\Sigma}(X)_{ds}$, $l \in T_{\mathsf{P}\Sigma}(X)_{l\text{-}ds}$, *if* $\mathrm{DSP} \vdash t \xrightarrow{l}_P t'$ *then* $\mathrm{DSP} \not\vdash t \xrightarrow{l}_{NP} t'$, *where* $\vdash$ *denotes provability in the Birkhoff system of Prop. 5 extended with the axioms* $x \xrightarrow{y}_P z \supset x \xrightarrow{y} z$ *and* $x \xrightarrow{y}_{NP} z \supset x \xrightarrow{y} z$ *(asserting that* $\longrightarrow$ *is the union of* $\longrightarrow_P$ *and of* $\longrightarrow_{NP}$*),*

then there exists an initial element in $Mod_{\mathsf{P}}(\mathrm{DSP})$. $\square$

Under the assumptions of Prop. 8, the above extended Birkhoff system is now sound and complete w.r.t. atoms and P-models.

4 An Example of Use of P

We want to specify using P the following vending machine.

The vending machine accepts tokens and is able to provide different drinks.

The price of each drink is fixed and equal to the value of the token. Each time a token is inserted, the current credit is incremented by the token value. It is possible to insert several tokens in a row, but when the "maximum credit" has been reached, the extra tokens have to be returned.

Subsequently to the selection of a drink, if the credit is positive, then the vending machine starts to fulfill the request, supplying cup and drink. It is permitted to do a selection only when the vending machine is not fulfilling any request; the selection can be modified if vending machine has not started the procedure to supply a drink.

The vending machine is switched on and off by the external world and could be repaired.

Requirement specification

If we use the ADS/P framework to specify the requirements on the vending machine, then we determine the class of all P-algebras (LTSs) formally modelling acceptable realizations of the vending machine by giving a P-requirement specification as follows. Recall that a P-requirement specification has loose semantics, i.e. its semantics is the class of its models, which are P-algebras.

We first determine which are the interactions of the vending machine with the external world (the labels of the LTS modelling the machine) and give some operations for representing them; afterwards we give the operations and predicates on the vending machine intermediate states (LTS states) and their properties using 1^{st} order formulae; and finally the properties on the vending machine activity (LTS transitions) by a set of P formulae, split in several groups.

For a generic ADS we have found that it is convenient to group together all formulae about a specific kind of interactions; since we are specifying a reactive system, in this way, we are guided by its interactions; for the P variant, the formulae about a kind of interactions are further split into:

– about normal execution,
– about abnormal execution,
– recovery from abnormal executions,
– general properties, that do not depend on the normal/abnormal distinction.

Here, for lack of room, we do not give the complete specification of the requirement of the vending machine, which can be found in [7], but report only some fragments of the various parts. In this case we have just one dynamic sort *vending_machine*.

Interactions

The interactions of the vending machine with the external world are:

** A token is introduced in the vending machine
$TOKEN$: *l-vending_machine*
** A drink is selected
$SELECT$: *drink* $\rightarrow$ *l-vending_machine*
** To supply a drink
$SUPPLY$: *drink* $\rightarrow$ *l-vending_machine*
** To break down
$BREAK\_DOWN$: *l-vending_machine*
** To return a token
$RETURN\_TOKEN$: *l-vending_machine*
** To be reset
$RESET$: *l-vending_machine*
** To be repaired
$REPAIR$: *l-vending_machine*
....

States

There are the following operations and predicates on the vending machine intermediate states.

** Return the value of the current credit
Credit: *vending_machine* $\rightarrow$ *nat*
** Return the current price of the drinks
Price: *vending_machine* $\rightarrow$ *nat*
** The price of drinks is always less than 1000 (an axiom)
$Price(vm) \leq 1000$
** Check if the vending machine is serving (a predicate)
Serving: *vending_machine*
** Check if the vending machine is broken
Broken: *vending_machine*
** Check if the vending machine is off
Off: *vending_machine*
** Check if a drink has been selected
Selection: *vending_machine* $\times$ *drink*
....

Activity

- *TOKEN*

`(general cases)`

If a token may be (normally/abnormally) introduced in the vending machine, then the vending machine is not broken

$(\exists\, vm' \,.\, vm \xrightarrow{TOKEN} vm') \supset \neg\, Broken(vm)$

Notice that $\exists\, vm' \,.\, vm \xrightarrow{TOKEN} vm'$ is equivalent to $\bigtriangledown(vm, \langle TOKEN \rangle)$

`(normal cases)`

If a token is normally introduced in the vending machine, then the vending machine is not serving, the credit will be incremented, and if nothing wrong occurs it will surely supply a drink

$vm \xrightarrow{TOKEN}_P vm' \supset$

$\neg\, Serving(vm) \wedge Credit(vm') = Credit(vm) + Price(vm) \wedge$

$\underset{\text{Safe}}{\bigtriangleup}(vm', \Diamond \langle \exists d \,.\, l = SUPPLY(d) \rangle)$

If a token is normally introduced in the vending machine infinite times, then, if nothing wrong occurs, the vending machine will surely supply a drink infinite times

$\underset{\text{Safe}}{\bigtriangleup}(vm, (\Box \Diamond \langle TOKEN \rangle) \supset (\Box \Diamond \langle \exists\, d \,.\, l = SUPPLY(d) \rangle))$

`(recovery)`

If a token is abnormally introduced in the vending machine, then the vending machine will return the token along a recovered path

$vm \xrightarrow{TOKEN}_{NP} vm' \supset \underset{\text{Rec}}{\bigtriangledown}(vm', \Diamond \langle RETURN\_TOKEN \rangle_P)$

- *SELECT*

`(general cases)`

If the vending machine is not off, then any drink may be selected

$\neg Off(vm) \supset \exists\, vm' \,.\, vm \xrightarrow{SELECT(d)} vm'$

`(normal cases)`

If a drink d is normally selected, then neither the vending machine has a valid selection, nor is serving nor is broken, and the current selection will be d; moreover if the credit is more than the price, if nothing wrong occurs, the vending machine will supply d

$vm \xrightarrow{SELECT(d)}_P vm' \supset$

$\nexists d' \,.\, Selection(vm, d') \wedge \neg Broken(vm) \wedge \neg Serving(vm) \wedge$

$Selection(vm', d) \wedge (Credit(vm) \geq Price(vm) \supset \underset{\text{Safe}}{\bigtriangleup}(vm', \Diamond \langle SUPPLY(d) \rangle))$

`(abnormal cases)`

If a drink is abnormally selected, then the current selection is not changed

$vm \xrightarrow{SELECT(d)}_{NP} vm' \supset (\forall d \,.\, Selection(vm', d) \equiv Selection(vm, d))$

- *RESET*

`(general cases)`

The vending machine may always be reset

$\exists vm' \,.\, vm \xrightarrow{RESET} vm'$

`(normal cases)`
If the vending machine is normally reset, then the credit is null, it has a current selection and is not broken; after there will be no current selection

$vm \xrightarrow{RESET}_{P} vm' \supset$
$Credit(vm) = 0 \wedge \exists d.Selection(vm, d) \wedge \neg Broken(vm) \wedge \neg \exists d'.Selection(vm', d')$

`(abnormal cases)`
If the vending machine is abnormally reset, then the current selection is not changed

$vm \xrightarrow{RESET}_{NP} vm' \supset (\forall d \,.\, Selection(vm', d) \equiv Selection(vm, d))$

....

Design specification

If we use the ADS/P framework to specify the design of the vending machine, then we determine one P-algebra (LTS) formally modelling the designed vending machine by giving a P-design specification as follows. Recall that a P-design specification has the initial semantics, i.e. its semantics is the (unique up to isomorphism) initial element in the class of its models.

We first determine which are the interactions of the vending machine with the external world and give some operations for representing them; afterwards we give operations to represent the states, one operation for each kind of states; and finally the axioms defining the transitions of the LTS (the vending machine activity) by using conditional formulae satisfying the conditions of Prop. 8.

For ADSs we have found that it is convenient to group together the axioms defining the activity starting from states of a specific kind (i.e. all those defining transitions with starting state of such kind); for the P variant for each state kind we define also which are the non-permitted transitions starting from states of such a kind and the related recovery activity using a special syntactic format.

Here, as before for the requirement specification, for lack of room, we do not give the complete specification, which can be found in [7], but report only some fragments of the various parts.

Labels As for the requirement specification, plus τ: *l-vending_machine* which corresponds to a null interaction with the external world (i.e. internal activity).

States The vending machine may be in one of the following states.

`* *` Waiting characterized by the credit
  Wait: *nat* $\to$ *vending_machine*
`* *` After having received a valid selection characterized by a drink
  Selected: *drink* $\to$ *vending_machine*
`* *` Serving a request characterized by the credit and a drink
  Serv: *nat* $\times$ *drink* $\to$ *vending_machine*
`* *` Off
  Off: *nat* $\to$ *vending_machine*
`* *` Broken but on (a constant, i.e. a zero-ary operation)
  Br: *vending_machine*

....

We use also a special operation

Rec: l-$vending\_machine$ $vending\_machine$ $\rightarrow$ $vending\_machine$

describing the states in which something of abnormal has happened and that a recovery is necessary: $Rec(l, s)$ represents a recovery state starting from which the only transition is permitted and has label l and final state s. So, in the activity part we have implicitly the axiom $Rec(l, s) \xrightarrow{l}_P s$.

Activity

- **Wait**

`(permitted activity)`

If the credit is less than the maximum, then a token may normally be introduced in the waiting vending machine and the credit is incremented (here the price is a constant)

$c < Max\_Credit \supset Wait(c) \xrightarrow{TOKEN}_P Wait(c + Price)$

If the credit is more than the price, then a drink may normally be selected in the waiting vending machine which passes to serve

$c \geq Price \supset Wait(c) \xrightarrow{SELECT(d)}_P Serv(c, d)$

If the credit is null, then a drink may normally be selected in the waiting vending machine which passes to selected

$Wait(0) \xrightarrow{SELECT(d)}_P Selected(d)$

The waiting vending machine may normally be switched off, keeping recorded the current credit

$Wait(c) \xrightarrow{SWITCH\_OFF}_P Off(c)$

`(nonpermitted activity)`

If the credit is maximum, the introduction of a token is abnormal. Recovery consists of returning the token without changing the credit

$c \geq Max\_Credit \supset Wait(c) \xrightarrow{TOKEN}_{NP} Rec(RETURN\_TOKEN, Wait(c))$

A reset is abnormal in the waiting status. Recovery consists of an internal action restoring the waiting status

$Wait(c) \xrightarrow{RESET}_{NP} Rec(\tau, Wait(c))$

A breakdown is abnormal. Recovery consists of being repaired restoring the waiting status with no credit

$Wait(c) \xrightarrow{BREAK\_DOWN}_{NP} Rec(REPAIR, Wait(0))$

- **Selected**

`(permitted activity)`

The selected vending machine may normally be reset passing to wait

$Selected(d) \xrightarrow{RESET}_P Wait(0)$

A token may normally be introduced into the selected vending machine, which passes to serve, while the credit becomes the price of a drink

$Selected(d) \xrightarrow{TOKEN}_P Serv(Price, d)$

`(nonpermitted activity)`
A selection is abnormal. Recovery consists of an internal action restoring the selected status

$$Selected(d) \xrightarrow{SELECT(d')}_{NP} Rec(\tau, Selected(d))$$

. . . .

In this example the recovery activity is simple and immediately performed, but we can also consider more complex cases where several violations may occur and we have to establish a policy for deciding which violations must recovered first and whether some violation imposes to forget other ones.

It is worthwhile to say that the above design specification has been proved correct w.r.t. the above requirements, see [7].

5 Concluding Remarks and Further Work

We have presented a way to extend the ADSs logic formalism for the specification of dynamic systems with the deontic concept of permission. This attempt seems worthwhile both from a theoretical and a methodological point of view.

The resulting formalism P is sound, indeed it turns out to be an institution (see [7]) and it is really an extension of ADSs (an ADS is a P-specification in which all transitions are permitted, i.e. $\longrightarrow \; = \; \longrightarrow_P$). The usual theoretical tools equipping ADSs have been uplifted to P specifications, as requirement and design specifications, a notion of when a specification correctly implements another one, sound/complete deductive systems and so on. Moreover, P does not have the paradoxes which frequently bother other deontic logics.

From a methodological point of view, we have that the requirement and design specifications of a dynamic system can be split in parts concerning respectively: – the normal cases without considering any failure of the system; – the possible failures (internal, due e.g. to the breaking of parts of the system, or external, due e.g. to wrong usages from the outside world, recall that dynamic systems are open and that usually assume that external users behave following some norms); – the recovery, if any, from each failure.

Moreover, we can give a syntactic format explicitly corresponding to such modular structure to the specifications; and that can help to write, to read and to modify complex specifications.

With regard to the future development of this work we plan to go along the following lines.

Obviously, we will have to analyse how to build up a complete deontic formalism, i.e. a formalism dealing with permission and obligation at the same time.

A formalism for obligation (called O, see [6]) has already been developed, whose main features are as follows. First of all, obligations are elements of a *regulation* that may be associated with the states of the LTSs. Such a regulation gives a picture of what the system *ought to do* starting from a particular state. For homogeneity with permission, obligations are applied over transitions, described by particular terms having form $\{\lambda x, l, y \,.\, \phi(x, l, y)\}$ which denotes the set of transitions $\{x \xrightarrow{l} y \mid \phi(x, l, y)\}$.

Using different operators, we may express immediate/non-immediate obligations and associate with them revocation as well as deadline conditions. For example, the atom $Reg(s, \mathbf{Ob}(tr)\mathbf{Rev}(tr_R)\mathbf{Before}(tr_B))$ says that in the regulation of s there is the non-immediate obligation to perform a transition in tr before one in tr_B, unless one in tr_R has been performed before one in tr_B (tr, tr_R, tr_B are terms representing sets of transitions). The deadline condition allows to associate the concept of *violation* with non-immediate obligations too.

The fact that O explicitly handles violations makes possible to describe which is the recovery from particular non-ideal behaviours (behaviours not satisfying obligations introduced along them), simply by putting new obligations on the violation states.

For example, in the vending machine specification we can have the axiom

$$vm \xrightarrow{TOKEN} vm' \wedge \neg Wait(vm) \supset$$
$$Reg(vm', \mathbf{Ob^{imm}}(\{\lambda x, l, y \,.\, l = RETURN\_TOKEN\}))$$

saying the vending machine has the immediate obligation to return a token inserted when it is not waiting (i.e. to perform a transition with label $RETURN\_TOKEN$).

If the token is not returned (i.e. the above obligation has been violated), the vending machine ought to be eventually either overhauled or repaired. This is expressed by the formula

$$\mathbf{Vi}(vm, \mathbf{Ob^{imm}}(\{\lambda x, l, y \,.\, l = RETURN\_TOKEN\})) \supset$$
$$Reg(vm, \mathbf{Ob}(\{\lambda x, l, y \,.\, l = OVERHAUL \vee l = REPAIR\}))$$

As for the permission case, from the description of a regulation and the analysis of the system behaviours, we may associate some qualifications with states: a state is *ideal*, *acceptable* and *faulty* iff it satisfies all its own obligations, respectively, in all behaviours, in some behaviour and in no behaviour. We can also make these qualifications relative to a particular obligation. It is worthwhile to note that, each time we require a state to be ideal (acceptable) w.r.t. an obligation, we force the state to satisfy it in every (at least one) future behaviour.

A complete formalism might be constructed starting from P by adding the above constructors for expressing obligations and regulations, as well as the relative qualifications. In the complete formalism there are two different and completely independent *structures* (permission and obligation), that are used to describe different aspects of ideal and real behaviour of a system. In such a way, we do not impose any particular relationship between permissions and obligations (e.g. we can oblige a system to perform non-permitted transitions).

However, it is possible, and from a methodological point of view advisable, to consider a subformalism within which, e.g., obligations are only on permitted transitions; thus where it is only possible to express that a transition ought to be performed, immediately or eventually, but in a permitted way. This is really a subformalism, since the new combinators for "permitted obligations" may be expressed by particular formulae of the complete formalism, as e.g.

$$Reg(vm, \mathbf{Ob^{imm}}(\{\lambda x, l, y \,.\, \phi \wedge x \xrightarrow{l}_P y\})).$$

A relevant point for the proposed formalisms based on deontic concepts is to experiment them on real case studies, to find out which are most common forms of recovery and to determine a class of formats large enough to cover them; so that handling of such cases becomes simple and natural.

The temporal logic T for ADSs of Sect. 2.3 has been extended to express safety and liveness properties concerning not just single states and single transitions but more complex activities; such activities are represented by whole sequences of transitions and are called abstract events, see [19, 2]. We want to extend both permission and obligation to act on abstract events, as it has been done in a different context in [15, 11].

The specification method founded over basic ADSs offers a friendly specification language, methodological guidelines for writing specifications, how to associate with formal specifications corresponding informal specifications (see [4, 3]), software tools to help to asses the correctness of a specification; moreover, there are also graphical tools helping in the use of such formalism, which avoid to write a long list of axioms that could become very complicated especially for not specialized users. We need to extend them to cover the new specifications using the deontic concepts.

Finally we want just to mention a different use of deontic concepts as a basis for specification formalisms (see e.g. [13]), worthwhile of further investigations, that consists of using the concepts of permission and obligation to restrict the activities of a system and to formalize properties it must absolutely satisfy. Such use is not properly deontic, since there is no distinction between real and ideal behaviour, but it gives us a very natural way to formulate the requirements of a system (see [6] for an example of such formalisms).

A Algebras with Predicates

Here we summarize the main definitions and facts about *algebras with predicates*, see [12].

A *many-sorted predicate signature*, shortly, a *signature*, (just a 1st order language) is a triple $\Sigma = (S, OP, PR)$, where

- S is a set (the set of the *sorts*);
- OP is a family of sets: $\{OP_{w,s}\}_{w \in S^*, s \in S}$; *operation symbols*
- PR is a family of sets: $\{PR_w\}_{w \in S^+}$; *predicate symbols*

A Σ*-algebra*, just a 1st order structure, is a triple

$\mathrm{A} = (\{\mathrm{A}_s\}_{s \in S}, \{Op^{\mathrm{A}}\}_{Op \in OP}, \{Pr^{\mathrm{A}}\}_{Pr \in PR})$

consisting of the *carriers*, the *interpretation of the operation symbols* and the *interpretation of the predicate symbols*. More precisely:

- if $s \in S$, then A_s is a set;
- if $Op: s_1 \times \ldots \times s_n \to s$, then $Op^{\mathrm{A}}: \mathrm{A}_{s_1} \times \ldots \times \mathrm{A}_{s_n} \to \mathrm{A}_s$ is a function;
- if $Pr: s_1 \times \ldots \times s_n$, then $Pr^{\mathrm{A}} \subseteq \mathrm{A}_{s_1} \times \ldots \times \mathrm{A}_{s_n}$.

Usually we write $Pr^{\mathrm{A}}(a_1, \ldots, a_n)$ instead of $(a_1, \ldots, a_n) \in Pr^{\mathrm{A}}$.

Given a signature Σ with set of sorts S, a *sort assignment* on Σ is an S-indexed family of sets $X = \{X_s\}_{s \in S}$.

Given a sort assignment X, the *term algebra* $\mathrm{T}_\Sigma(X)$ is the Σ-algebra defined as usual.

In this paper we assume that algebras have *nonempty carriers* (as this applies to term algebras as well, we have an implicit assumption on signatures: that they contain "enough constants symbols").

If A is a Σ-algebra, a *variable evaluation* $V: X \rightarrow \mathrm{A}$ is a sort-respecting assignment of values in A to *all* the variables in X. If $t \in \mathrm{T}_\Sigma(X)$, the *interpretation of* t *in* A *w.r.t.* V is denoted by $t^{\mathrm{A},V}$ and given as usual.

If A and B are Σ-algebras, a *homomorphism* h from A into B, written $h: \mathrm{A} \rightarrow \mathrm{B}$, is a family of total functions $h = \{h_s: \mathrm{A}_s \rightarrow \mathrm{B}_s\}_{s \in S}$ s.t.:

- for all $Op \in OP$: $h_s(Op^{\mathrm{A}}(a_1, \dots, a_n)) = Op^{\mathrm{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$;
- for all $Pr \in PR$: if $Pr^{\mathrm{A}}(a_1, \dots, a_n)$, then $Pr^{\mathrm{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$.

Given a class of Σ-algebras $\mathcal{C}$, an algebra I is *initial* in $\mathcal{C}$ iff $\mathrm{I} \in \mathcal{C}$ and for all $\mathrm{A} \in \mathcal{C}$ there is a unique homomorphism $h^{\mathrm{A}}: \mathrm{I} \rightarrow \mathrm{A}$; notice that the initial element is unique up to isomorphism.

Proposition 9. *If* I *is initial in* $\mathcal{C}$*, then for all ground terms* $t_1, \dots, t_n$ *and all predicates* $Pr \in PR$*:*

- $\mathrm{I} \models t_1 = t_2$ *iff for all* $\mathrm{A} \in \mathcal{C}$*:* $\mathrm{A} \models t_1 = t_2$*;*
- $\mathrm{I} \models Pr(t_1, \dots, t_n)$ *iff for all* $\mathrm{A} \in \mathcal{C}$*:* $\mathrm{A} \models Pr(t_1, \dots, t_n)$. $\square$

References

1. E. Astesiano and G. Reggio. SMoLCS-Driven Concurrent Calculi. In H. Ehrig, R. Kowalski, G. Levi, and U. Montanari, editors, *Proc. TAPSOFT'87, Vol. 1*, number 249 in Lecture Notes in Computer Science, pages 169–201. Springer Verlag, Berlin, 1987.
2. E. Astesiano and G. Reggio. Specifying Reactive Systems by Abstract Events. In *Proc. of Seventh International Workshop on Software Specification and Design (IWSSD-7)*. IEEE Computer Society, Los Alamitos, CA, 1993.
3. E. Astesiano and G. Reggio. A Case Study in Friendly Specifications of Concurrent Systems (Lamport & Broy's Specification Problem Presented at the Dagstuhl Seminar "Specification and Refinement of Reactive Systems – A Case Study"). Technical Report DISI–TR–94–21, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994.
4. E. Astesiano and G. Reggio. Formally-Driven Friendly Specifications of Concurrent Systems: A Two-Rail Approach. Technical Report DISI–TR–94–20, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994. Presented at ICSE'17-Workshop on Formal Methods, Seattle April 1995.
5. E. Astesiano and G. Reggio. Algebraic Dynamic Specifications: An Outline. Technical Report DISI–TR–95–08, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1995.
6. E. Coscia. Utilizzo di Concetti Deontici nella Specifica Formale di Sistemi Dinamici. Master Thesis. In Italian, 1995.
7. E. Coscia and G. Reggio. Deontic Concepts in the Specification of Dynamic Systems: the Permission Case. Technical Report DISI–TR–95–14, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1995. Available by anonymous ftp at `ftp.disi.unige.it`, directory `pub/reggio`.
8. G. Costa and G. Reggio. Abstract Dynamic Data Types: a Temporal Logic Approach. In A. Tarlecki, editor, *Proc. MFCS'91*, number 520 in Lecture Notes in Computer Science, pages 103–112. Springer Verlag, Berlin, 1991.
9. G. Costa and G. Reggio. Specification of Abstract Dynamic DataTypes: A Temporal Logic Approach. *T.C.S.*, 1996. Avaible by anonymous ftp at `ftp.disi.unige.it`, directory `/pub/reggio`.

10. P. D'Altan, J.-J.Ch. Meyer, and R.J. Wieringa. An Integrated Framework for Ought-to-Be and Ought-to-Do Constraints. Technical Report IR-342, Faculty of Mathematics and Computer Science, Vrije University, Amsterdam, 1993.
11. J. Fiadeiro and T. Maibaum. Temporal Reasoning over Deontic Specification. *J. Logic Computation*, 1(3):357–395, 1991.
12. J. Goguen and J. Meseguer. Models and Equality for Logic Programming. In *Proc. TAPSOFT'87, Vol. 2*, number 250 in Lecture Notes in Computer Science, pages 1–22. Springer Verlag, Berlin, 1987.
13. A.J.I. Jones and M. Sergot. On the Characterization of Law and Computer Systems: The Normative Systems Perspective. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.
14. S.J.H. Kent, T.S.E. Maibaum, and W.J. Quirk. Formally Specifying Temporal Constraints and Error Recovery. In *Proc. of International Symposium on Requirements Engineering RE'93*. IEEE Computer Society, Los Alamitos, CA, 1993.
15. S. Khosla. *System Specification: a Deontic Approach*. PhD thesis, Imperial College, London, 1988.
16. J.-J. Ch. Meyer, F.P.M. Dignum, and R.J. Wieringa. The Paradoxes of Deontic Logic Revisited: A Computer Science Perspective. Technical Report UU-CS-1994-38, Department of Computer Science, Utrecht University, 1994.
17. J.-J. Ch. Meyer, F.P.M. Dignum, and R.J. Wieringa. A Solution to the Free Choice Paradox by Contextually Permitted Actions. *Studia Logica*, 1995. To be published.
18. J.-J.Ch. Meyer and R.J. Wieringa, editors. *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.
19. G. Reggio. Event Logic for Specifying Abstract Dynamic Data Types. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification*, number 655 in Lecture Notes in Computer Science, pages 292–309. Springer Verlag, Berlin, 1993.
20. G. Reggio and E. Crivelli. Specification of a Hydroelectric Power Station: Revised Tool-Checked Version. Technical Report DISI–TR–94–17, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1994.
21. G. Reggio and V. Filippi. Specification of a High-Voltage Substation: Revised Tool-Checked Version. Technical Report DISI-TR-95-09, Dipartimento di Informatica e Scienze dell'Informazione – Università di Genova, Italy, 1995.
22. M. Wirsing. Algebraic Specifications. In J. van Leeuwen, editor, *Handbook of Theoret. Comput. Sci.*, volume B, pages 675–788. Elsevier, 1990.

Acknowledgement We warmly thank Roel Wieringa and Josè Fiadeiro, for having introduced us to the realm of deontic logic and the referees for their careful reading and helpful comments.

This article was processed using the LaTeX macro package with LLNCS style