F. Dignum, H. Weigand and E. Verharen. Meeting the deadline: on the formal specification of temporal deontic constraints. In Z.W. Ras and M. Michalewicz, editors, *Foundations of Intelligent Systems (LNAI 1079)*, pages 243--252. Springer-Verlag, 1996.

Meeting the deadline: on the formal specification of temporal deontic constraints

F.Dignum[1], H. Weigand and E. Verharen[2]

[1] Fac. of Maths. & Comp. Sc., Eindhoven University of Technology, The Netherlands
[2] Infolab, Tilburg University, The Netherlands

Abstract. In this paper, we describe a temporal deontic logic that facilitates reasoning about obligations and deadlines. The logic is an extension of deontic dynamic logic, in which only immediate obligations can be specified. In our extension, we can also specify that an obligation starts at a certain time or event, that it must be done immediately, as soon as possible, before a deadline, or periodically. A practical application area are intelligent agents that must be able to reason about their agendas.

1 Introduction

It is not very difficult to develop a program that checks whether deadlines are met. The main idea is to wait until the deadline has passed, which usely can be checked easily, and then check whether a certain action has taken place. However, many difficulties arise when one tries to transform this procedural account of deadlines into a formal one. In an intelligent system (or agent) one would like to be able to reason about this type of constraints in order to check whether they can be fulfilled at all. This holds especially for combinations of different deadlines. Constraints can also be used to influence the behaviour of the agent. The combination of deadlines can be used to plan the actions of an agent. Of course this is only possible if the system has some formal description (besides a procedural one) of the deadlines.

A related issue is the occasion that a certain deadline is not met. What should be the consequences of the failure to meet a deadline? If one sees the failure to meet a deadline as a constraint violation, in some systems this would mean that the system reaches an inconsistent state. In these systems (most database systems currently in use) the constraints have to be fulfilled at any moment in time. Of course this can easily be enforced for static constraints. Any update of the system that violates a constraint is rejected. However, deadlines are constraints with a fundamental different nature. Whether a deadline is met depends on an action that must have taken place. In case this action only depends on the system itself one might force the system to perform the action before the deadline is reached. In this case the deadline would be used as part of the planning system of the intelligent system. The deadlines would not have to be checked afterwards because they would be met by default (if possible of course).

Several problems can arise using this method. First of all it is not clear at what time the action should be planned. As soon as possible or just before the deadline? This problem can be aggrevated by deadlines of which it is not known beforehand when they will be reached. E.g. place an order before the stock falls below a certain level. It is not known at what point in time the stock falls below that level. We argue that the enforcement of the deadline and the planning problem are two separate issues and the enforcement of deadlines should not be implemented by a planning procedure. Of course we do acknowledge that the deadlines influence the planning of the actions of the intelligent system.

A second problem that arises with the enforcement of deadlines is that the system is not always capable of enforcing the performance of a certain action. E.g. upon delivery of the product the customer has to pay the bill within 30 days. The system can base its plans on the fact that the customer has paid within 30 days, but it has no way to enforce this payment (directly). This shows that deadlines cannot always be enforced. Therefore a system should not reach a state of inconsistency whenever a deadline is not met. Rather it should arrive at a state in which it is clear that a deadline has passed, but other (corrective) actions are still possible. (In case of the customer the system could send a reminder or a court order for payment).

A last problem that we like to mention is the case where no specific deadline has been set. A certain action should take place "as soon as possible". E.g. after an accident has been reported the ambulance has to go to the place of the accident as soon as possible. However, it might be that the ambulance first has to deliver another patient at the hospital or that the accident is not very serious and the ambulance does not switch on its siren. In these cases there is not a definite point in time where one can check whether the action has been performed or not.

In this paper we aim at formally describing deadlines such that it is possible to reason about them. One important requirement is that it must be possible to violate the deadline without the system entering an inconsistent state. In order to fulfill this requirement we use a form of dynamic deontic logic. In this logic it is possible to state that a certain action should take place without getting an inconsistency when the action does not take place.

We will show that the following types of deadlines can be described in the logical formalism in a natural and concise form.

1. Deadlines concerning a specific time relative to the current moment.
2. Deadlines that depend on a certain state that may be established by an external action. (e.g. order before the stock is to low).
3. Flexible deadlines, i.e. obligations that should be fulfilled "as soon as possible"
4. Periodic deadlines. I.e. payment of employees each month.

The rest of this paper is organized as follows. First we will introduce the dynamic deontic logic that is used to describe the deadlines. In section 3 we will

describe the types of deadlines given above in the dynamic deontic logic (using some concrete examples). Section 4 contains some conclusions and directions for further research.

2 A logic of actions and norms

We now proceed with the definition of a set of *formulas* with which we can describe the behaviour of (interpreted) (trans)actions. This language is a variant of *dynamic logic* ([10]), and was first used for this purpose in [12]. In the present paper we add a few "new" formulas to this language. They are the ones defined in points (5), (6) and (7) below. The formulas defined in (5) involve a variant on the standard dynamic logic definition of the consequence of (trans)actions. The formulas defined in (6) all involve some type of temporal operations on actions. Allthough more approaches exist that combine temporal and deontic logics (e.g. [13, 11, 9]) these approaches tend to express the deontic concepts in terms of the temporal operators. In this paper we take a different approach. We actually "add" the temporal operators to the deontic logic that is used as a basis. The formulas defined in (7) define the "classical" deontic formulas as introduced by v.Wright in [14, 1]. Finally, the formulas defined in (8) introduce a preference relation between actions. They state which action is preferred to be performed at a certain time.

We assume a fixed set *Prop* of atomic propositions and sets Act and *Tract* of action expressions and transaction expressions respectively (see below). The set *Form* of formulas with typical elements ϕ and ψ is given by the following BNF:

$$\begin{aligned}\phi ::- \; & p|\phi \wedge \psi|\neg\phi|[\beta]\phi| \ll \beta \gg \phi| \\ & PAST(\alpha, i)|PREV(\alpha)|O(\phi)|PREFER(\alpha, \alpha')\end{aligned}$$

with p a proposition, α, α' actions and β a transaction (defined below).
Note: Other propositional connectives such as $\vee$ and $\rightarrow$ are assumed to be introduced as the usual abbreviations. Also the special proposition $false$ is introduced as the abbreviation of $p \wedge \neg p$ for some $p \in Prop$. The informal meaning of $[\beta]\phi$ is "doing β necessarily leads to a state where ϕ holds". $\ll \beta \gg \phi$ means that after performing the transaction denoted by β the formula ϕ holds and it does not hold before β is completely performed. The meaning of $PAST(\alpha, i)$ is that α has actually been performed i steps ago. The meaning of $PREV(\alpha)$ is "the present state is actually reached by performing α". Note that we make a difference between the possible ways that the state can be reached and the way it actually *is* reached. The informal meaning of $O(\phi)$ is that ϕ should be the case in the present state. The last type of formulas defined above indicate that a certain action α is preferred to performing α'.

The semantics of the formulas in *Form* is given in two stages. First we will give the syntax and intuitive semantics of the transaction expressions, which we will subsequently use in section 2.2 to define the semantics of the formulas.

2.1 Transaction expressions

We start out by giving a definition of *transaction expressions*, which we shall typically denote α, possibly with subscripts. To this end we assume a set At of *atomic action expressions* that are typically denoted by $\underline{a}, \underline{b}, \ldots$. Furthermore, we assume special action expressions **any** and **fail** denoting "don't care what happens" and "failure", respectively.

Definition 2.1 The set Act of action expressions is given by the following BNF:

$$\alpha :: -\underline{a} | \mathbf{any} | \mathbf{fail} | \alpha_1 + \alpha_2 | \alpha_1 \& \alpha_2 | \overline{\alpha}$$

The set *Tract* of transaction expressions is given by the following BNF:

$$\beta :: -\underline{a} | \mathbf{any} | \mathbf{fail} | \beta_1 + \beta_2 | \beta_1 \& \beta_2 | \overline{\beta} | \beta_1 ; \beta_2$$

Note that the definition of Act is almost the same as for *Tract* except that we do not allow for sequences of actions. To keep the logic as simple as possible the temporal operators only reach over action expressions and not over transaction expressions.

The semantics of (trans)action expressions has two components. The first component is an algebra of uninterpreted actions (called a *uniform* semantics elsewhere [3]), which allows us to interpret equalities between action expressions without taking their effect into account. In the algebraic semantics, each action expression will be interpreted as a choice over possible steps. Due to a lack of space we refer to [7] for a formal account of this part. Most important is that we have a formal semantics for the negation of (trans)actions as well. Informally the semantics of a (trans)action α ($[\![\alpha]\!]$) is given as a set of *traces*.

Definition 2.2 A trace is a finite or infinite sequence $S_1 S_2 \ldots S_n \ldots$ of steps. ϵ stands for the empty trace.

The second component of the semantics is a state-transition semantics of action expressions where we define the effect of steps on the state of the world. Most notable here is that the effect of any action can always be a set of new states. (One for each trace in its semantics)

We define the duration of action expressions as follows.

Definition 2.3 The **duration** of α is defined as $dur(\alpha) = dur([\![\alpha]\!])$.

2.2 Semantics of formulas

Having defined the action expressions within the formulas, we can now give the semantics of formulas in *Form* by means of the notion of a Kripke structure $\mathcal{M} = (\mathcal{A}, \Sigma, \pi, R_{\mathcal{A}}, \leq, R_O)$.
Σ is a set of states (worlds).
$\mathcal{A}$ is a finite set of events.
π is a truth assignment function to the atomic propositions relative to a state:

π is a function $\Sigma \rightarrow (Prop \rightarrow \{tt, ff\})$, where tt and ff denote truth and falsehood, respectively. Thus, for $p \in Prop$, $\pi(\sigma)(p) = tt$ means that the atomic proposition p is true in state σ.

The accessibility relation $R_{\mathcal{A}}$ specifies how transactions can change states. The relation $R_{\mathcal{A}}$ is defined as follows: $R_{\mathcal{A}} = \{R_t | t \ a \ trace\}$, reflecting that R_t is the relevant entity.
$\leq$ is a function $(\Sigma \times A^*) \times (\Sigma \times A^*) \rightarrow \{tt, ff\}$. The function indicates for two state/history pairs which of the two is preferred.
In this paper we only use the preference relation to indicate a preference relation between actions. We do not give a logic for the preference relation itself. However, one might intuitively think that an action α is preferred over an action β if it leads to states in which less constraints are violated or the violations are considered less harmfull(i.e. which are more ideal in a deontic sense). For a thorough treatment of this type of logic we refer to [4]. Here we take the preference relation to be primitive.

R_O is the deontic relation that with respect to a state σ reached by trace γ indicates the ideal situation consisting of state σ' and trace γ'. R_O resembles the classical deontic relation in modal interpretations, except that we do not consider only states, but pairs of states and traces. We assume the relation R_O to be serial. I.e. for every world σ and trace γ there exists at least one pair (σ', γ') such that $R_O((\sigma, \gamma)(\sigma', \gamma'))$ holds.

We now give the interpretation of formulas in *Form* in Kripke structures. We interpret formulas with respect to a structure $\mathcal{M}$ and a pair (σ, γ)

Definition 2.4 Given $\mathcal{M} = (\mathcal{A}, \Sigma, \pi, R_{\mathcal{A}}, R_O, I)$ as above and (σ, γ), we define:

1. $(\mathcal{M}, (\sigma, \gamma)) \models p \Longleftrightarrow \pi(\sigma)(p) = tt$ (for $p \in Prop$)
2. $(\mathcal{M}, (\sigma, \gamma)) \models \phi_1 \wedge \phi_2 \Longleftrightarrow (\mathcal{M}, (\sigma, \gamma)) \models \phi_1$ and $(\mathcal{M}, (\sigma, \gamma)) \models \phi_2$
3. $(\mathcal{M}, (\sigma, \gamma)) \models \neg\phi \Longleftrightarrow$ not $(\mathcal{M}, (\sigma, \gamma)) \models \phi$
4. $(\mathcal{M}, (\sigma, \gamma)) \models [\alpha]\phi \Longleftrightarrow \forall t \in [\![\alpha]\!] \forall \sigma' \in \Sigma [R_t(\sigma, \sigma') \Rightarrow (\mathcal{M}, (\sigma', \gamma \circ S)) \models \phi]$
5. $(\mathcal{M}, (\sigma, \gamma)) \models \ll \alpha \gg \phi \Longleftrightarrow \exists (\sigma', \gamma') : \gamma' = \gamma \circ t \wedge t \in [\![\alpha]\!] \wedge (\mathcal{M}, (\sigma', \gamma')) \models$
   $\phi \wedge \neg \exists \alpha_1 \exists \alpha_2 : (\alpha_1; \alpha_2 =_{\mathcal{D}} \alpha) \wedge \exists (\sigma' \gamma') : \gamma' = \gamma \circ t \wedge t \in [\![\alpha_1]\!] \wedge (\mathcal{M}, (\sigma', \gamma')) \models \phi$
6. $(\mathcal{M}, (\sigma, \gamma)) \models O(\phi) \Longleftrightarrow \forall (\sigma', \gamma') [R_O((\sigma, \gamma), (\sigma', \gamma')) \Rightarrow (\mathcal{M}, (\sigma', \gamma')) \models \phi]$
7. $(\mathcal{M}, (\sigma, \gamma)) \models PREV(\alpha) \Longleftrightarrow \exists S \in [\![\alpha]\!], \gamma' [\gamma = \gamma' \circ S]$
8. $(\mathcal{M}, (\sigma, \gamma)) \models PAST(\alpha, i) \Longleftrightarrow \exists \sigma' \in \Sigma \exists \gamma', \gamma'' [\gamma = \gamma' \circ \gamma'' \wedge$
   $dur(\gamma'') = i \wedge (\mathcal{M}, (\sigma', \gamma')) \models PREV(\alpha)]$
9. $(\mathcal{M}, (\sigma, \gamma)) \models PREFER(\alpha_1, \alpha_2) \Longleftrightarrow \forall t \in [\![\alpha_2]\!] (R_t(\sigma, \sigma_2) \longrightarrow$
   $(\exists t' \in [\![\alpha_1]\!] (R_{t'}(\sigma, \sigma_1) \wedge (\sigma_1, \gamma \circ t') \leq (\sigma_2, \gamma \circ t))))$
10. ϕ is *valid* w.r.t. model $\mathcal{M} = (\mathcal{A}, \Sigma, \pi, R_{\mathcal{A}}, R_O, I)$, notation $\mathcal{M} \models \phi$,
    if $(\mathcal{M}, (\sigma, \gamma)) \models \phi$ for all $\sigma \in \Sigma$ and γ.

11. ϕ is *valid*, notation $\models \phi$, if ϕ is valid w.r.t. all models $\mathcal{M}$ of the form considered above.

The first four definitions are quite standard and we will not explain them any further here. In (5) we define the fact that the condition ϕ becomes true for the first time after performing the (complete) transaction denoted by α.
The definition of the static obligation involves both the state and the trace (and *not* just the state). In this way, we can express that the circumstances described by $PREV(\alpha)$, for example, are obligatory. For example, it might be obligatory to have just done the action indicated by α. This means that the history (i.e. the trace) of an ideal world might differ from the history of the present world. We will use this feature to define obligations on actions shortly.
It should be noted that using the semantic definition of $[\mathbf{any}]\phi$ we can express the usual temporal operators over static formulas as given in e.g. [8]. Points (7) and (8) define extra temporal operators reaching over action expressions!
Point (9) defines what it means that one action is preferred over another action.

Before we give a definition of the deontic operators in the next section, we will introduce a helpful operator. This operator indicates that a formula ϕ is true as soon as a formula ψ becomes true. It is defined formally as follows:

Definition 2.5

$$\phi \textbf{ when } \psi \equiv \ll \gamma \gg \psi \longrightarrow [\gamma](\psi \rightarrow \phi)$$

2.3 Obligations and deadlines

Using the above definitions we can now introduce the deontic operators over actions. We will introduce one general type of obligations: the obligation with deadlines. Using this general type, we will then define some special types of obligations that are often used to describe all types of deadlines.

Definition 2.6 $O(\phi < \alpha < \psi) \equiv O(\alpha < \psi)$ **when** ϕ
with $O(\alpha < \psi) \equiv \ll \gamma \gg \psi \wedge dur(\gamma) = n \longrightarrow [\gamma](\psi \rightarrow O(\exists i : 0 \leq i < n : PAST(\alpha, i)))$

The most general form $O(\phi < \alpha < \psi)$ stands for the fact that α should be performed after ϕ has become true and before ψ has become true. Intuitively $O(\alpha < \psi)$ stands for the fact that α should be performed before ψ holds true. I.e. if ψ becomes true (sometimes) for the first time after γ then it is obliged that α has been performed in the course of γ (in the last n steps).

The first specialisation of the general obligation is made with respect to the begin and end conditions of the period in which the action should be performed. The definition in the general case is somewhat complicated because whether these conditions hold true might depend on the type of (trans)action that is performed. In the case that we take the conditions to be purely temporal they do not depend on the transaction performed anymore. We distinguish between

relative and absolute time conditions. For the absolute time conditions we can introduce a special variable $time$. We assume all actions to take equal time and the length of an action defines the basic unit of time. Using this axiom we define the general obligation with pure temporal deadlines as follows:

Definition 2.7 $O(now + temp_1 < \alpha < now + temp_1 + temp_2) \equiv (time = now \wedge [\mathbf{any}^n]time = now + temp_1) \wedge [\mathbf{any}^n][\mathbf{any}^m]time = now + temp_1 + temp_2) \longrightarrow [\mathbf{any}^{n+m}]O(\exists 0 \leq i < m : PAST(\alpha, i))$
and $O(\alpha < now + temp_2) \equiv (time = now \wedge [\mathbf{any}^m]time = now + temp_2) \longrightarrow [\mathbf{any}^m]O(\exists 0 \leq i < m : PAST(\alpha, i))$

The next specialisation of the general case is in fact the type of dynamic obligation as it is used in most dynamic deontic logics. It is the "immediate" obligation, which means that the action should be performed as the next action. The immediate obligation is defined as follows:

Definition 2.8 $O!(\alpha) \equiv O(\alpha < now + 1)$

From the definitions the following equivalence can easily be proven:

Proposition 2.9 $O!(\alpha) \equiv [\mathbf{any}]O(PREV(\alpha))$

So, α is obligated if, whatever I do now, it will be true immediately afterwards that I was just previously obligated to do α. This means that if I do $\overline{\alpha}$, I reach a state where a violation occurs.

With the general type of obligation with deadlines it is also possible to describe an obligation that has to be fulfilled as soon as possible. This obligation is interpreted as meaning that the action should be performed as soon as no other actions with a higher "preference" are performed. The definition is as follows:

Definition 2.10 $O?(\alpha) \equiv O(true < \alpha < PREV(\beta) \wedge PREFER(\alpha, \beta))$

This obligation can be used when no strict deadline is given, but we want the action to be performed at some time. It resembles the "liveness" property as described in [9], except that the obligated action cannot be postponed indefinite. It has to be performed before an action with lesser importance is performed.

The last type of obligation that we will describe is the periodic obligation. This obligation returns every time a certain condition holds true and should be fulfilled before another condition holds true. E.g. an order should be placed after the stock of computers has fallen below 15 and before the level dropped below 5. Although this seems the same as the general obligation described above it is a bit different. The condition that the stock falls below a certain level will be true periodically (one hopes) and every time this happens an order for replenishment should be made. The periodic obligation is described as follows:

Definition 2.11
$PO(\phi < \alpha < \psi) \equiv \forall n : dur(\gamma) = n \longrightarrow [\gamma](O(\alpha < \psi) \vee justdone(\alpha))$
with
$justdone(\alpha) \equiv (\exists 0 \leq i < n - k : PAST(\alpha, i)) \wedge \gamma = \beta_1; \beta_2 \wedge$
$dur(\beta_1) = k \wedge [\beta_1]\phi \wedge (\forall \beta' : \beta' = \beta_1; \beta_3 \wedge dur(\beta') < n \rightarrow \neg[\beta']\phi))$

$justdone(\alpha)$ states that α has been done after the last time that ϕ became true. The definition of $PO(\phi < \alpha < \psi)$ states that (from now on) it is always obligated to do α before ψ holds true except when α has been "justdone".

3 Modelling deadlines

In this section we will model the types of deadlines given in the introduction within the logical framework developed in the previous section using some concrete example for each type of deadline.

> When a CS student is enrolled in the university then he has to pass the exam in "introduction to programming" within the first year.

This example is modeled as follows:

$$\forall p : PREV(Enroll(p, CS) \rightarrow O(Pass(p, IP) < now + year)$$

I.e. if $Enroll(p, CS)$ has just been done then there is an obligation to perform the action $Pass(p, IP)$ between "now" and a year time. We assume that *year* stands for an integer that indicates how many times an action should be performed to advance the absolute time with one year. Although parameterized actions were not explicitly introduced in this paper, we use them in the examples in order to get a more realistic representation. The formal introduction of parameterized actions can be found in [6].

The second example shows some combinations of different types of deadlines.

> After the stock of computers has fallen below 10 an order should be made before the stock is less than 6. If an order has been made the delivery should follow within 5 days. If the delivery is not made in time a reminder should be sent. After the receipt of the goods payment should be affectuated within 30 days.

This example is modelled by the following formulas:

(1) $O(PREV(fall - stock(Computers < 10)) < Order < stock(Computers) < 6)$
(2) $PREV(Order) \rightarrow O(Delivery < now + 5 * day)$
(3) $(PREV(Order) \wedge [\mathbf{any}^{5*day}](\neg\exists_{0 \leq i < 5*day} PAST(Delivery, i))) \longrightarrow [\mathbf{any}^{5*day}]O!(Send(reminder))$
(4) $PREV(Receipt) \rightarrow O(Pay < now + 30 * day)$

The third formula is a typical example of how the violation of an obligation triggers another obligation. This is very natural, because the violation of an obligation should lead to some rectifying action, which is usually an obligation as well.

The next example illustrates an obligation that should be fulfilled "as soon as possible".

After a customer has phoned to register a failing central heating system (during the winter) a mechanic should try to repair it as soon as possible, but at least within 24 hours. (From a contract between a service company and a client).

This is an example of having an obligation to perform an action as soon as possible. In this case it might be that the service company is very busy and got several calls at the same time. In that case it is not possible to go to all clients at the same time. However, if the mechanic goes to all the clients one after the other we would say he fulfilled the obligation of the service company. The above example can be modeled as follows:

$$PREV(Report(ch)) \longrightarrow (O?(Try - repair(mechanic, ch)) \wedge \\ O(Try - repair(mechanic, ch) < now + 24 * hour))$$

The last example illustrates the use of periodic obligations.

The employees of the company have to be paid their salaries between the 25th and 30th day of each month.

The above example can be modelled very simple as follows:

$$PO(monthday(time) = 25 < Pay(salary, emp) < monthday(time) = 30)$$

where *monthday* is a function that returns the day of the month given an absolute point in time.

4 Conclusions

We have shown in this paper how deadlines can be modelled using a type of dynamic deontic logic. Deadlines play an important role in flexible transactions. In situations where several systems have to cooperate deadlines are a means to specify expectations of the behaviour of the other parties. E.g. if a company delivers a product it expects a payment of the customer within a certain time.

Actions and transactions are necessary ingredients in the specification of deadlines. First of all deadlines are always specified on actions. I.e. every deadline indicates that a certain *action* is expected to take place. Secondly the deadline may be dependent on the (trans)actions that are performed. E.g. You have to pay the rent before you buy a new car. These considerations indicate that a formal specification of deadlines should involve a formal specification of actions. We have chosen a form of dynamic logic to incorporate the actions into the specification language.

A second important property of deadlines is that they are not always kept. In the case that keeping a deadline depends on an action from another system (or person) it is not possible to enforce the deadline. Therefore we should use a formalism that allows the violation of the deadline without getting in an inconsistent state. This requirement is fulfilled by the incorporation of deontic logic into the specification language.

The use of a logic as specification language enables the system to reason about the deadlines. Deadlines can be combined and inconsistent deadlines (deadlines that cannot be kept jointly) can be detected.

The present work also opens some areas for further research. In particular the temporal aspects of the language are rather primitive. We assume that all actions take the same amount of time, which, of course, is not very realistic. A second area for further research is the influence of the deadlines on the (planning of) actions of the system.

Acknowledgements We would like to thank the anonymous referees for giving some very usefull remarks that put this research into a wider context.

References

1. L. Åqvist. Deontic logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic II*, pages 605–714. Reidel, 1984.
2. J.C.M. Baeten and W.P. Weijland. Process Algebra. Cambridge University Press, 1990.
3. J.W. de Bakker, J.N Kok, J.-J.Ch. Meyer, E.-R. Olderog, and J.I. Zucker. Contrasting themes in the semantics of imperative concurrency. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency: Overviews and Tutorials*, pages 51–121. LNCS 224 Springer, Berlin, 1986.
4. C. Boutilier Toward a Logic for Qualitative Decision Theory. In JonDoyle, Erik Sandewall and Pietro Torasso (eds.), *Principles of Knowledge Representation and Reasoning, proceedings of the fourth international conference*, pages 75–86, 1994, Morgan Kaufmann Publishers, San Francisco, California.
5. M. Broy. A theory for nondeterminism, parallelism, communication and concurrency. In *Theoretical Computer Science*, vol.45, pages 1–62, 1986.
6. F. Dignum and J.-J.Ch. Meyer. Negations of transactions and their use in the specification of dynamic and deontic integrity constraints. In M. Kwiatkowska, M.W. Shields, and R.M. Thomas, editors, *Semantics for Concurrency, Leicester 1990*, pages 61–80, Springer, Berlin, 1990.
7. Contextual permission. a solution to the free choice paradox. In A. Jones and M. Sergot, editors, *Second International Workshop on Deontic Logic in Computer Science*, pages 107–135, Oslo, 1994. Tano A.S.
8. E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995-1072, North-Holland, Amsterdam, 1989.
9. J. Fiadeiro and T. Maibaum. Temporal Reasoning over Deontic Specification. In *Journal of Logic and Computation*, 1 (3), 1991.
10. D. Harel. First Order Dynamic Logic. LNCS 68 Springer, 1979.
11. J.F. Horty. Combining Agency and Obligation. In M. Brown and J. Carmo (eds.), *Deontic Logic, Agency and Normative Systems*, pages 98-122, Springer-Verlag, Berlin, 1996.
12. J.-J.Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. In *Notre Dame*, vol.29, pages 109–136, 1988.
13. R. Thomason. Deontic Logic as founded on tense logic. In R. Hilpinen, editor, *New Studies in Deontic Logic*, pages 165-176, D.Reidel Publishing Company, 1981.
14. G.H. von Wright. Deontic logic. In *Mind*, vol.60, pages 1–15, 1951.