A formal specification of deadlines using dynamic deontic logic

Eindhoven University of Technology
Department of Mathematics and Computing Science

A Formal Specification of Deadlines using Dynamic Deontic Logic

by

F. Dignum, H. Weigand and E. Verharen

96/09

ISSN 0926-4515

Computing Science Report 96/09
Eindhoven, April 1996

A Formal Specification of Deadlines using Dynamic Deontic Logic.

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

[1] Fac. of Maths. & Comp. Sc., Eindhoven University of Technology
P.O.box 513, 5600 MB Eindhoven, The Netherlands,
tel.+31-40-473705, email: dignum@win.tue.nl
[2] Tilburg University, Infolab,
P.O.box 90153, 5000 LE Tilburg, The Netherlands,
tel.+31-13-662806, email: h.weigand@kub.nl

Abstract. Intelligent agents have an agenda that is monitored continuously to decide what action is to be performed. Formally, an agenda is a set of deontic temporal constraints. Deontic, since the agenda specifies what the agent should do. Temporal, since the obligation is usually to be performed before a certain deadline, or as soon as possible. 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.

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 examples of deadlines can be described in the logical formalism in a natural and concise form.

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

> 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 effectuated within 30 days.

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).

The employees of the company have to be paid their salaries between the 25th and 30th day of 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. Especially the semantics of (trans)actions and of different types of obligations are discussed. In section 3 we will describe the examples given above in the dynamic deontic logic. In section 4 we will introduce an equivalent axiomatization in the form of frame axioms. Section 5 contains some conclusions and directions for further research. We also briefly indicate how deontic temporal constraints are used in Agent Specification.

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. [14, 11, 9]) these approaches tend to express the deontic concept s 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 [15, 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 is then the smallest set closed under:

(1). $\mathit{Prop} \subseteq \mathit{Form}$

(2). $\phi_1, \phi_2 \in \mathit{Form} \Longrightarrow \phi_1 \wedge \phi_2 \in \mathit{Form}$

(3). $\phi \in \mathit{Form} \Longrightarrow \neg\phi \in \mathit{Form}$

(4). $\alpha \in \mathit{Tract}, \phi \in \mathit{Form} \Longrightarrow [\alpha]\phi \in \mathit{Form}$

(5). $\alpha \in \mathit{Tract}, \phi \in \mathit{Form} \Longrightarrow \ll \alpha \gg \phi \in \mathit{Form}$

(6). $\alpha \in \mathit{Act} \Longrightarrow PAST(\alpha, i), PREV(\alpha) \in \mathit{Form}$

(7). $\phi \in \mathit{Form} \Longrightarrow O(\phi) \in \mathit{Form}$

(8). $\alpha_1, \alpha_2 \in \mathit{Act} \Longrightarrow PREFER(\alpha_1, \alpha_2) \in \mathit{Form}$

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 $p \in \mathit{Prop}$. The informal meaning of $[\alpha]\phi$ is "doing α necessarily leads to a state where ϕ holds". $\ll \alpha \gg \phi$ means that after performing the transaction denoted by α the formula ϕ holds and it does not hold before α 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. In our semantics, we will therefore consider state-history pairs. The informal meaning of $O(\phi)$ is that ϕ should be the case in the present state. Because we include histories in our semantic structure, we can express that the current state should, ideally, have been reached by another history. We introduce the other deontic operators using the usual abbreviations:

- $F(\phi)$ abbreviates $O(\neg\phi)$
- $P(\phi)$ abbreviates $\neg F(\phi)$

The last type of formulas defined above (in (8)) indicate that a certain action α_1 is preferred to performing α_2.

The semantics of the formulas in *Form* is given in two stages. First we will give the syntax and 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 and their semantics

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 *Tract* of transaction expressions is given as the smallest set closed under:

(i). $At \cup \{\mathbf{any}, \mathbf{fail}\} \subseteq \mathit{Tract}$

(ii). $\alpha_1, \alpha_2 \in \mathit{Tract} \Longrightarrow \alpha_1 + \alpha_2 \in \mathit{Tract}$

(iii). $\alpha_1, \alpha_2 \in \mathit{Tract} \Longrightarrow \alpha_1 \& \alpha_2 \in \mathit{Tract}$

(iv). $\alpha \in \mathit{Tract} \Longrightarrow \overline{\alpha} \in \mathit{Tract}$

(v). $\alpha_1, \alpha_2 \in \mathit{Tract} \Longrightarrow \alpha_1 ; \alpha_2 \in \mathit{Tract}$

We define the set of action expressions *Act* to be the smallest set closed under:

(i). $\mathit{At} \cup \{\mathbf{any}, \mathbf{fail}\} \subseteq \mathit{Act}$

(ii). $\alpha_1, \alpha_2 \in \mathit{Act} \Longrightarrow \alpha_1 + \alpha_2 \in \mathit{Act}$

(iii). $\alpha_1, \alpha_2 \in \mathit{Act} \Longrightarrow \alpha_1 \& \alpha_2 \in \mathit{Act}$

(iv). $\alpha \in \mathit{Act} \Longrightarrow \overline{\alpha} \in \mathit{Act}$

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 is given in two stages. First we define 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. Next, we give a state-transition semantics of action expressions in which we define the effect of steps on the state of the world.

Algebraic action semantics. With every atomic action expression $\underline{a} \in \mathit{Act}$, we associate an event a in a given class $\mathcal{A}$ of events, with typical elements $a, b, c....$ Different atomic action expressions are associated with different events in $\mathcal{A}$. Events are the semantical entities on which we shall base our interpretation of action expressions. We further assume a special event δ, which is not an element of $\mathcal{A}$, called failure (comparable to deadlock in process algebra ([2]). The relation between an action expression $\underline{a} \in \mathit{Act}$ and the associated event $a \in \mathcal{A}$ is more involved than just interpreting $\underline{a}$ as a. We shall interpret atomic action expressions $\underline{a} \in \mathit{Act}$ in a more sophisticated way, which we call "open": the meaning of an atomic action expression $\underline{a} \in \mathit{Act}$ will be the event $a \in \mathcal{A}$ corresponding with it, in combination with any other subset of the events in $\mathcal{A}$. Thus $\underline{a}$ expresses that a occurs, but it leaves open which other events occur simultaneously (in the same step) with a. The intuitive motivation for this is that if we say that an event a occurs, we do not mean that nothing else occurs in the world.

Definition 2.2

1. The set $\{\delta\}$ is a step.

2. Every non-empty finite subset of $\mathcal{A}$ is a step. The powerset of non-empty finite subsets of $\mathcal{A}$ will be denoted by $\mathcal{P}^+(\mathcal{A})$.

Notation: In concrete cases we write the sets indicating steps with square brackets, in order to distinguish them easily from other sets that we will use. So, the step consisting of δ is written as $[\delta]$ and the step consisting of the events a and b is written as $\begin{bmatrix} a \\ b \end{bmatrix}$.

The above definition prevents the simultaneous execution of the special event δ with other events, because it is not in $\mathcal{A}$. This is necessary, because it is not possible to perform an event and at the same time have a deadlock.

To denote the subsequent execution of actions we make use of sequences of steps. These sequences can be finite or infinite. We will call these sequences of steps "traces" conform the terminology used in the semantics of concurrent programming [5]. The definition of a trace is given as follows:

Definition 2.3 A trace is a finite or infinite sequence $S_1 S_2 \ldots S_n \ldots$ of steps. ϵ stands for the empty trace.
Only the last step of a trace may be $[\delta]$.
The number of steps in a trace t is called the *length* or *duration* of t, denoted by $dur(t)$. $dur(\epsilon) = 0$

Notation: We use $t, t_1, t_2, \ldots, t', \ldots$ to denote traces. We use A^* to denote the set of all traces that can be formed from $\mathcal{A}$.

Definition 2.4 The domain D for our model of transaction expressions from *Tract* is the collection of sets of traces.
An element of D is called a choice set and is denoted with $T, T_1, \ldots.$

The use of choice sets as elements of the domain indicates the inherent non-determinism of the performance of the actions. Only when the semantics of a transaction expression consists of a choice set with one element will the transaction be deterministic.

Just as for traces we can define the length of a choice set (which will indicate the length of the transaction expression):

Definition 2.5 The *length* or *duration* of a choice set T, denoted by $dur(T)$ is defined as:

$$dur(T) = max\{dur(t) | t \in T\}$$

Below, we interpret transaction expressions in terms of choice sets. In order to do this we define the semantical counterparts of the syntactic operators in Tract ($\cup$, $+$, $^-$ and ";").
Before we give the definitions of these operators, we will define some helpful functions. We start with the definition of prefixes of traces.

Definition 2.6
$pref(t) = \{t' | t' \circ t'' = t\}$

Note that ϵ is an element of the *pref* of any trace. The $\circ$ operator denotes concatenation of traces and is defined formally in definition 9 below.
The next function defines the longest common prefix of two traces.

Definition 2.7 Let $t_1 = S_1 \dots S_n \dots$ and $t_2 = S'_1 \dots S'_m \dots$
Then $maxpref(t_1, t_2)$ is the longest trace t such that $t \in pref(t_1)$ and $t \in pref(t_2)$.

Note that if $S_1 \neq S'_1$, $maxpref(t_1, t_2) = \epsilon$.

Finally we define an operator (T^δ) on choice sets, which removes traces ending in $[\delta]$. These traces are only removed if there is another trace that is the same but with $[\delta]$ replaced by another trace.

Definition 2.8 Let T be a choice set then

$$T^\delta = T \backslash \{t | t = t'[\delta] \wedge \exists t'' \in T : t'' \neq t \wedge t' \in pref(t'')\}$$

The operator T^δ is closely related to what is called "failure removal" in [3]. The idea is that failure is avoided when possible, i.e. when there is a non-failing alternative. In [5], this is called *angelic* nondeterminism.

We will now define the semantical counterparts of each of the syntactic operators. We will start with the simplest, the ";". The semantical counterpart of this operator is the concatenation of choice sets (representing the semantics of the transactions that are performed in sequence).

The definition of the concatenation operator is given in the following:

Definition 2.9 1. Let $t = S_1 \dots S_n$ and $t' = S'_1 \dots S'_m$ be two traces (possibly infinite) then

$$t \circ t' = \begin{cases} S_1 \dots S_n & \text{if } S_n = [\delta] \\ S_1 \dots S_n S'_1 \dots S'_m & \text{if } S_n \neq [\delta] \end{cases}$$

If t is an infinite trace, then $t \circ t' = t$ for any trace t'.
$t \circ \epsilon = \epsilon \circ t = t$.

2. Let T and T' be choice sets, then $T \circ T'$ is defined as the choice set $\{t \circ t' | t \in T, t' \in T'\}$.

Note: $T \circ \{\epsilon\} = \{\epsilon\} \circ T = T$ and $\{[\delta]\} \circ T = \{[\delta]\}$ and $T \circ \{[\delta]\} = \{t \circ [\delta] | t \in T\}$.

For the parallel operator & we use a set-intersection $\Cap$, which is almost the same as the normal set-intersection, except that a trace can appear in the intersection not only if it appears in both sets, but also if it appears in one set and the other set contains a prefix of it. The definition assures that if two transactions are compatible, then the length of the transaction that results from performing them simultaneously is equal to the length of the longest transaction.

Definition 2.10

1. Let t and t' be traces:

$$t \mathbin{\dot\cap} t' = \begin{cases} t & \text{if } maxpref(t,t') = t' \\ t' & \text{if } maxpref(t,t') = t \\ maxpref(t,t') \circ [\delta] & \text{otherwise} \end{cases}$$

2. Let $T, T' \in \mathcal{D}$:

$$T \mathbin{\dot\cap} T' = (\cup\{t \mathbin{\dot\cap} t' | t \in T', t' \in T'\})^{\delta}$$

The semantical counterpart of the choice operator is defined as follows:

Definition 2.11 For $T, T' \in \mathcal{D}$:

$$T \mathbin{\dot\cup} T' = ((T \cup T') \backslash (\cup\{t \mathbin{\dot\cap} t' | t \in T, t' \in T' \wedge t \neq t'\}))^{\delta}$$

The above definition states that the choice between two choice sets is the union of those two choice sets minus some type of intersection. However, it is not the actual intersection of the two choice sets that is subtracted, but those traces that only appear in one set and have a prefix in the other set.
This complicated definition is needed to secure intuitive equalities like:

$$\alpha_1 + (\alpha_1; \alpha_2) = \alpha_1$$

Finally, we define the semantic counterpart of the negation operator.

Definition 2.12 The definition of "$\sim$" is given as follows:

1. For a step S,

$$S^{\sim} = \wp^{+}(\mathcal{A}) \backslash \{S\}$$

2. For a non-empty trace $t = S_1 \ldots S_m \ldots$

$$t^{\sim} = \bigcup_{n \leq dur(t)} S_1 \circ \cdots \circ S_n{}^{\sim}$$

3. For a non-empty set $T \in \mathcal{D}$

$$T^{\sim} = \dot\cap_{S \in T} S^{\sim}$$

That is, for a step the negation just yields the set-theoretic complement of $\{S\}$ with respect to $\wp^{+}(\mathcal{A})$. The negation of a trace consists of all the traces that start with a prefix and end with the negation of the step following that prefix. The negation of a choice set T is the "special" intersection of the sets(!) of the negations of all the traces contained in T.

We can now give the algebraic semantics of action expressions:

Definition 2.13 The semantic function $[\![]\!] \in Act \rightarrow \mathcal{D}$ is given by:

$$\begin{array}{l} [\![\underline{a}]\!] = \{S \in \wp^{+}(\mathcal{A}) | a \in S\} \\ [\![\alpha_1 + \alpha_2]\!] = [\![\alpha_1]\!] \uplus [\![\alpha_2]\!] \\ [\![\alpha_1 \& \alpha_2]\!] = [\![\alpha_1]\!] \Cap [\![\alpha_2]\!] \\ [\![\alpha_1 ; \alpha_2]\!] = [\![\alpha_1]\!] \circ [\![\alpha_2]\!] \\ [\![\overline{\alpha}]\!] = [\![\alpha]\!]^{\sim} \\ [\![\mathbf{fail}]\!] = \{[\delta]\} \\ [\![\mathbf{any}]\!] = \wp^{+}(\mathcal{A}) \end{array}$$

The first clause of the above definition expresses that the meaning of the action expression $\underline{a}$ is exactly as we have described informally before: it is the set of steps that contain the event a, representing a choice between all (simultaneous) performances of sets of events which at least contain the event a, so that the performance of a is guaranteed but also other events may happen simultaneously. The meaning of the action expression **fail** is comparable to a deadlock. The only event that can be performed is δ. The action expression **any** is the complement of **fail**. It stands for a choice of any possible combination of events.

Finally we define duration, equality and implication between action expressions.

Definition 2.14 The **duration** of α is defined as $dur(\alpha) = dur([\![\alpha]\!])$.
Action expressions α_1 and α_2 are *equal*, written $\alpha_1 =_{\mathcal{D}} \alpha_2$,
iff $[\![\alpha_1]\!] = [\![\alpha_2]\!]$. α_1 *involves* or *implies* α_2, written $\alpha_1 > \alpha_2$, iff $[\![\alpha_1]\!] \subseteq [\![\alpha_2]\!]$.

State-transition action semantics. To get a state-transition semantics, we postulate what effects events have in terms of state transformations (we do this relative to a set Σ of states). We assume that there is a function $eff : \mathcal{A} \rightarrow (\Sigma \rightarrow \Sigma)$, such that $eff(a)$ is a function from states to states. (For simplicity, we assume events to be deterministic. Elsewhere, we show how nondeterministic events can be incorporated [12].) Two events are called *compatible* if their joint effect is independent of the order in which they occur.

Definition 2.15 Let $S = [a_1, ..., a_n] \subseteq \mathcal{A}$ be a step consisting of pairwise compatible events. The accessibility relation $R_S \subseteq \Sigma \times \Sigma$ is defined as follows:

$$R_S(\sigma, \sigma') \Longleftrightarrow_{def} eff(a_1) \circ ... eff(a_n)(\sigma) = \sigma'$$

On the basis of the accessibility relation R_S we also define an accessibility relation based on traces.

Definition 2.16 Let t and t' be traces then:

$$R_{tot'}(\sigma, \sigma') \Longleftrightarrow_{def} \exists \sigma'' R_t(\sigma, \sigma'') \wedge R_{t'}(\sigma'', \sigma')$$

2.2 Semantics of formulas

Having defined the semantics of 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: $R_{\mathcal{A}} = \{R_t | t \in A^*\}$, reflecting that R_t is the relevant entity. We write, slightly abusing notation,

$$R_{\mathcal{A}}(\sigma, \sigma') \Longleftrightarrow \exists t : R_t(\sigma, \sigma')$$

$\leq$ 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.

The relation R_O relates state/history pairs (σ,γ) to state/history pairs (σ',γ'). The history of a state is expressed by a *trace* of the actions that went before. In process algebra, temporal logic and semantics of parallel programs traces are widely used to record what actions have taken place.

We would like to ensure that it is possible for a state to be reached by its history. Therefore we introduce the following relation between states and traces relative to a model $\mathcal{M}$:

Definition 2.17 For each model $\mathcal{M} = (\mathcal{A}, \Sigma, \pi, R_{\mathcal{A}}, R_O, I)$ and pair (σ, γ) with $\sigma \in \Sigma$ and γ a trace, $(\sigma, \gamma) \in Comp(\mathcal{M})$ iff

1. $\gamma = \epsilon$ or
2. $\gamma = \gamma' \circ S \wedge \exists \sigma' \in \Sigma[R_S(\sigma', \sigma) \wedge (\sigma', \gamma') \in Comp(\mathcal{M})]$

From now on we assume that all pairs $(\sigma, \gamma) \in Comp(\mathcal{M})$ unless it is stated otherwise.

R_O is the deontic relation that with respect to a state σ reached by history γ indicates the ideal situation consisting of state σ' and history 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 $(\sigma,\gamma) \in Comp(\mathcal{M})$

Definition 2.18 Given $\mathcal{M} = (\mathcal{A}, \Sigma, \pi, R_{\mathcal{A}}, R_O, I)$ as above and $(\sigma,\gamma) \in Comp(\mathcal{M})$, 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. $(\mathcal{M},(\sigma,\gamma)) \models O(\phi) \Longleftrightarrow \forall(\sigma',\gamma') \in Comp(\mathcal{M})[R_O((\sigma,\gamma),(\sigma',\gamma')) \Rightarrow (\mathcal{M},(\sigma',\gamma')) \models$ $\phi]$
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 $[\textbf{any}]\phi$ 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 2.19

$$\phi \mathbf{when} \psi \equiv \ll \gamma \gg \psi \longrightarrow [\gamma]\psi \rightarrow \phi$$

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 2.20 $O(\phi < \alpha < \psi) \equiv O(\alpha < \psi)\mathbf{when}\phi$
with $O(\alpha < \psi) \equiv \ll \gamma \gg \psi \wedge dur(\gamma) = n \longrightarrow [\gamma](\psi \rightarrow O(\exists i : 0 \leq i < n : PAST(\alpha, i)))$

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 performing the transaction denoted by γ then whenever ψ holds after γ it is obliged that α has been performed in the course of γ (in the last n steps).
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.

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*. The following axiom should hold for the values of this variable:
Axiom 2.21

$$\models time = k \longrightarrow [\mathbf{any}]time = k + 1$$

That is, 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 absolute temporal deadline as follows:

Definition 2.22 $O(temp_1 < \alpha < temp_2) \equiv [\mathbf{any}^n]temp_1 \longrightarrow [\mathbf{any}^n]O(\alpha < temp_2)$
with $O(\alpha < temp_2) \equiv [\mathbf{any}^m]temp_2 \longrightarrow [\mathbf{any}^m]O(\exists 0 \leq i < m : PAST(\alpha, i))$

For relative deadlines the definitions are as follows:

Definition 2.23 $O(now{+}temp_1 < \alpha < now{+}temp_1{+}temp_2) \equiv (time = now \wedge [\mathbf{any}^n]time = now + temp_1) \wedge [\mathbf{any}^n][\mathbf{any}^m]time = now + temp_1 + temp_2) \longrightarrow [\mathbf{any}^{n+m}]O(\exists 0 \leq i < m : PAST(\alpha, i))$
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 2.24 $O!(\alpha) \equiv O(\alpha < now + 1)$

From the definitions the following equivalence can easily be proven:

Proposition 2.25 $O!(\alpha) \equiv [\mathbf{any}]O(PREV(\alpha))$

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, indicated by the fact that in that state both $O(PREV(\alpha))$ and $PREV(\overline{\alpha})$ hold true.
Note that by definition the following formula is valid for all actions $\alpha \in Act$:

$$[\alpha]PREV(\alpha)$$

Therefore

$$O!(\alpha) \Longrightarrow [\overline{\alpha}](\neg PREV(\alpha) \wedge O(PREV(\alpha)))$$

For the special case of the immediate obligation we can now define the other deontic operators for prohibition and permission in terms of the obligation:

- $F!(\alpha) \equiv O!(\overline{\alpha})$
- $P!(\alpha) \equiv \neg F!(\alpha)$

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 2.26 $O?(\alpha) \equiv O(true < \alpha < PREV(\beta) \wedge PREFER(\alpha, \beta))$

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 2.27 $PO(\phi < \alpha < \psi) \equiv \forall n : dur(\gamma) = n \longrightarrow [\gamma](O(\alpha < \psi) \vee justdone(\alpha))$
with
$justdone(\alpha) \equiv (\exists 0 \leq i < n - k : PAST(\alpha, i)) \wedge \gamma =_{\mathcal{D}} \beta_1; \beta_2 \wedge dur(\beta_1) = k \wedge [\beta_1]\phi \wedge (\forall \beta' : \beta' =_{\mathcal{D}} \beta_1; \delta \wedge dur(\beta') < n \rightarrow \neg[\beta']\phi))$

$justdone(\alpha)$ 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 (the examples)

In this section we will model the examples given in the introduction within the logical framework developed in the previous section.

> 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 Frame axioms

Instead of definition 20 above, it is also possible to provide frame axioms that specify whether the obligation is carried over to the next world or not. In this section, we will compare the two approaches.

An obligation persists until it is satisfied or violated. It is satisfied when the action is performed, and it is violated when at the time of evaluation (the deadline) it is not performed. If no deadline is given, the evaluation is assumed to be at the end of history (Judgment Day).

The frame axioms will be stated as propositions since they can be derived from the definitions given above. If α should be performed before ψ holds true then this obligation still holds if α is not performed and we have not reached the deadline ψ.

Proposition 4.1 *(i).* $\models (O(\alpha < \psi) \wedge \neg\psi) \longrightarrow [\overline{\alpha}]O(\alpha < \psi)$

(ii). $\models O(\alpha < \psi) \longrightarrow (\psi \rightarrow O(PREV(\alpha)))$

Proof:

According to definition 20 $O(\alpha < \psi)$ means that (traces in) transactions γ leading to ψ also lead to $O(PAST(\alpha, i))$. Now let $O(\alpha < \psi)$ be true at state s_1, and let s_2 be a state reached from s_1 via some action but not α. Then it is evident that all (traces in) transactions γ_1 from s_2 leading to ψ also lead to $O(PAST(\alpha, i))$, since each (trace in) γ_1 is a subtrace of some (trace in) γ starting from s_1. The only restrictions are (1) that α should be somewhere in γ_1, so it should not be done already before s_2; and (2) that γ was not empty already, i.e., ψ did already hold in s_1. These two restrictions are met by the conditional part of proposition (i).

Proposition (ii) also follows from definition 20. If ψ holds, it means that the γ in the definition is empty. The definition can be reduced then to $O((\exists i : 0 \leq i < n : PAST(\alpha, i)))$ for $n = 0$, which is equivalent to $PREV(\alpha)$.

Note that according to the proposition the obligation disappears when α is performed or when the deadline is reached. This does not mean that $O(\alpha < \psi)$ is false afterwards, but rather that it may hold or not. This is specified simply by saying nothing about this case. The most we can say is that the obligation will not always hold:

$$\models (O(\alpha < \psi) \wedge \neg\psi) \longrightarrow \neg([\alpha]O(\alpha < \psi))$$

Under a Closed World Assumption, this will be interpreted as a negation.

The two propositions can also be take together as follows:

$$\models (O(\alpha < \psi) \wedge \neg\psi) \longrightarrow [\overline{\alpha}](O(\alpha < \psi) \wedge (\psi \rightarrow O(PREV(\alpha))))$$

5 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.

Deontic temporal constraints can be used in the specification of contracts between agents ([13]). The specification of an agent includes the specification of the possible messages it can exchange with other agents. Certain combinations of messages, for example, a request followed by a commit, create an obligation for one agent with regard to the other. Sometimes, such obligations can be retracted again by means of cancel messages. The *contract* between two agents describes the possible messages and their effect, that is, the factual or deontic temporal constraints that are created or deleted by means of the messages.

Agents have an agenda that contains the actions they are supposed to perform at due time. The agent architecture assumes that the agenda is monitored continuously to decide what to do next. The formal meaning of the agenda is a set of deontic temporal constraints. In the context of an agent architecture, a set of deontic temporal constraints functions as a program.

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. A simple algorithm would be to plan the action whose deadline expires first as the first action to perform. However, from the OR research (and personal experience) we might deduce that this does not always lead to optimal (or even acceptable) plans. So, some more complicated planning algorithms are called for.

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 fo urth 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. F. Dignum, J.-J.Ch. Meyer and R. Wieringa. 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 Normat ive 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 Journal of Formal Logic*, vol.29, pages 109–136, 1988.
13. Y. Shoham Agent-oriented programming. *Artificial Intelligence 60* 1993, pp.51-92.
14. 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.
15. G.H. von Wright. Deontic logic. In *Mind*, vol.60, pages 1–15, 1951.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Komatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.

94/39	A. Blokhuis, T. Kloks	On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks, D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet, J.J. Vereijken	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse, M. Bijsterveld	Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma, J. Davies, R. Gerth, S. Graf, W. Janssen, B. Jonsson, S. Katz, G. Lowe, M. Poel, A. Pnueli, C. Rump, J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo, F. Kamareddine, R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo, F. Kamareddine, R. Nederpelt	On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group	Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten, J.A. Bergstra	Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers	A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks, D. Kratsch, H. Müller	Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek, R. Kuiper	Traces and Logic, p. 81
94/53	R. Gerth, R. Kuiper, D. Peled, W. Penczek	A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien	The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem, R. Bol, J.F. Groote	Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten, C. Verhoef	Concrete process algebra, p. 134.
95/04	J. Hidders	An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi	A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen, M.G.A. Verhoeven, H.M.M. ten Eikelder, E.H.L. Aarts	A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn, O.S. van Roosmalen	Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo	Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten, J.A. Bergstra	Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse, R. Verhoeven, O. Weber	Math∫pad: A System for On-Line Prepararation of Mathematical Documents, p. 15

95/11	R. Seljée	Deductive Database Systems and integrity constraint checking, p. 36.
95/12	S. Mauw and M. Reniers	Empty Interworkings and Refinement Semantics of Interworkings Revised, p. 19.
95/13	B.W. Watson and G. Zwaan	A taxonomy of sublinear multiple keyword pattern matching algorithms, p. 26.
95/14	A. Ponse, C. Verhoef, S.F.M. Vlijmen (eds.)	De proceedings: ACP'95, p.
95/15	P. Niebert and W. Penczek	On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.
95/16	D. Dams, O. Grumberg, R. Gerth	Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.
95/17	S. Mauw and E.A. van der Meulen	Specification of tools for Message Sequence Charts, p. 36.
95/18	F. Kamareddine and T. Laan	A Reflection on Russell's Ramified Types and Kripke's Hierarchy of Truths, p. 14.
95/19	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra with Abstraction, p. 15.
95/20	F. van Raamsdonk and P. Severi	On Normalisation, p. 33.
95/21	A. van Deursen	Axiomatizing Early and Late Input by Variable Elimination, p. 44.
95/22	B. Arnold, A. v. Deursen, M. Res	An Algebraic Specification of a Language for Describing Financial Products, p. 11.
95/23	W.M.P. van der Aalst	Petri net based scheduling, p. 20.
95/24	F.P.M. Dignum, W.P.M. Nuijten, L.M.A. Janssen	Solving a Time Tabling Problem by Constraint Satisfaction, p. 14.
95/25	L. Feijs	Synchronous Sequence Charts In Action, p. 36.
95/26	W.M.P. van der Aalst	A Class of Petri nets for modeling and analyzing business processes, p. 24.
95/27	P.D.V. van der Stok, J. van der Wal	Proceedings of the Real-Time Database Workshop, p. 106.
95/28	W. Fokkink, C. Verhoef	A Conservative Look at term Deduction Systems with Variable Binding, p. 29.
95/29	H. Jurjus	On Nesting of a Nonmonotonic Conditional, p. 14
95/30	J. Hidders, C. Hoskens, J. Paredaens	The Formal Model of a Pattern Browsing Technique, p.24.
95/31	P. Kelb, D. Dams and R. Gerth	Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions, p. 17.
95/32	W.M.P. van der Aalst	Handboek simulatie, p. 51.
95/33	J. Engelfriet and JJ. Vereijken	Context-Free Graph Grammars and Concatenation of Graphs, p. 35.
95/34	J. Zwanenburg	Record concatenation with intersection types, p. 46.
95/35	T. Basten and M. Voorhoeve	An algebraic semantics for hierarchical P/T Nets, p. 32.
96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/04	S. Mauw	Example specifications in phi-SDL.
96/05	T. Basten and W.M.P. v.d. Aalst	A Process-Algebraic Approach to Life-Cycle Inheritance Inheritance = Encapsulation + Abstraction, p. 15.
96/06	W.M.P. van der Aalst and T. Basten	Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07	M. Voorhoeve	Structural Petri Net Equivalence, p. 16.
96/08	A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk	OODB Support for WWW Applications: Disclosing the internal structure of Hyperdocuments, p. 14.