Actor-Oriented System Specification with Dynamic Logic

J.-J.Ch. Meyer
R.J. Wieringa

Department of Mathematics and Computer Science
Free University
De Boelelaan 1981a
1081 HV Amsterdam

uucp: jules@cs.vu.nl, roelw@cs.vu.nl

ABSTRACT

In this paper, we extend dynamic logic with the concept of an actor in order to be able to specify who takes the initiative of an action, who makes a choice, or who controls a synchronization of actions. We give two examples of application of this idea. First, we show how to generalize an approach taken up by De Nicola and Hennessy, who eliminate τ from CCS in favor of internal and external choice. We show that this generalization allows a more accurate specification of system behavior than is possible without it. Second, deontic logic has been used by several researchers as a system specification language. In the course of this application, a number of paradoxes of classical deontic logic have been resolved, except the paradox of free choice permission. We show that actors can be used to resolve this paradox as well.

Subject area: Specification of systems, combining different logics

1. Introduction

1.1. Internal choice and internal events

Milner [26] sketches an intuitive picture of black box M equipped with buttons that may or may not be blocked by M, and if not blocked, can be pushed by an observer O. If M has two buttons, a and b, that are unblocked, then O is in a position to choose whether to push one or the other button. Using CSP-like notation, the process executed is then $a + b$, where + stands for external choice. If, on the other hand, M chooses to blocks one or the other button, then the process executed is $a \oplus b$, where $\oplus$ stands for internal choice.

This vivid example can be generalized to the case of n actors for any $n>1$, by allowing any actor

to make a choice. The distinction between internal and external then loses its meaning, for we will not identify with any actor in the system. We will write $\iota:(x+y)$ for the process that actor ι makes a choice between processes x and y.

In CCS, internal choice between a and b is represented by $\tau a + \tau b$, where τ is any action initiated by M. τ is called the *internal* or *invisible* action. It is well-known that the axiomatization of τ is not very intuitive, and De Nicola and Hennessy [27] show how to eliminate τ from CCS in favor of the intuitively more pleasing internal and external choice of CSP [15]. We think τ contains two ideas that should be distinguished, connected to *initiative* and *visibility*. τ is any action that occurs on the initiative of M as well as any action that is invisible to O. Separating these two concepts and generalizing to the case of n actors, we will explicitly add initiative to any event and leave open the question to which actors an event occurrence is visible. Thus, $M:a$ is event a initiated by M, and $O:a$ is a initiated by O. For example, if a and b are the events of a light going on and off, respectively, then $M:(M:a + M:b)$ is the process in which M chooses to switch the light on or off. If, on the other hand a and b are the events of pushing on or the other button, then $M:(O:a + O:b)$ is the process in which M blocks a button and O then pushes one. Events initiated by one actor may be visible by others.

Note that $\tau a + \tau b$ states that M makes a choice by performing an invisible action. Choice is itself not seen as an action, so that a property of + is expressed by a property of the first event of the branches. This makes it difficult to interpret terms like $a + \tau b$, where it is hard to say whether choice is internal or external [27].

1.2. The paradox of free choice permission

Deontic logic is the logic of permissions, prohibitions, and obligations [1, 9, 18, 28]. Recently, deontic logic has been applied to the specification of software systems [8, 20, 21, 31, 22, 30]. Traditionally, deontic logic has been plagued by numerous paradoxes. Castañeda and von Wright [5, 32] have proposed that a number of these paradoxes can be resolved by distinguishing actions from states. This approach has been formalized in [24, 25] using dynamic logic [13] . The basic idea is to label the set of possible states as either forbidden or permitted, and to define any action that leads to a forbidden state as forbidden. Permission and obligation can then be defined in a standard way in terms of prohibition.

One paradox still remains, however, called the *paradox of free choice permission* [14, 19]. This is that the following formula is derivable ($P(a)$ says that event a is permitted):

(1) $\quad P(\text{buy chewing gum}) \rightarrow P(\text{buy chewing gum + shoot the president}).$

This paradox can be resolved using the distinction between internal and external choice [23]. Permission to do a means that there is a way of doing a that leads to a permitted state of the world. One reading of (1) is therefore intuitively plausible, viz. if there is a way to chew gum that leads to a permitted state, then there is a way to perform the process (buy chewing gum + shoot the president) that leads to a permitted world (viz. by performing the permitted way to chew gum). On the other hand, it is counterintuitive to conclude from P(chew gum) that I am permitted to choose between chewing gum and shooting the president. Thus, (2) is a formalization of our intuition and (3) is not:

(2) $\quad P(\iota_1:a_1) \rightarrow P(\iota_2:(\iota_1:a + \iota_1:b))$

(3)* $\quad P(\iota_1:a_1) \rightarrow P(\iota_1:(\iota_1:a + \iota_1:b))$

(2) says that there is a possibility that ι_2 makes the choice in such a way that a permitted world will

ensue after performing the chosen action. (3) makes the incorrect statement

that if I am permitted to do something, then I permitted to choose to do something else as well. The force of the permission $P(\iota_1{:}(\iota_1{:}a + \iota_1{:}b)$ is stronger than that of $P(\iota_2{:}(\iota_1{:}a + \iota_1{:}b)$, because in addition to saying that there is a possibility that ι_1 chooses a permitted action, it says that ι_1 is permitted to choose between the actions. In our system, (2) is a theorem and (3) is not. In fact, we have the theorems

(4) $P(\iota_2{:}(\iota_1{:}a + \iota_1{:}b)) \leftrightarrow P(\iota_1{:}a) \vee P(\iota_1{:}b)$

(5) $P(\iota_1{:}(\iota_1{:}a + \iota_1{:}b)) \rightarrow P(\iota_1{:}a) \wedge P(\iota_1{:}b)$,

which agrees with our intuitions. (5) blocks the paradox of free choice permission.

2. Dynamic logic with equality and action negation

We specify a system as a set of possible states which all contain an underlying abstract data type (ADT) as reduct. Events and processes will be specified as functions on the set of possible states. Thus, the system is a Kripke structure with multiple accessibility relationships, one for each event and process. More in detail, we model any system as shown in figure 1.

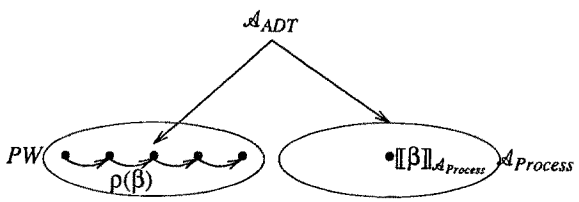

Figure 1.

Each world in the set *PW* of possible worlds contains the underlying ADT, called $\mathcal{A}_{ADT}$, as reduct. In addition, there is an algebra $\mathcal{A}_{Process}$ of processes, together with a function ρ which, for each process, yields a function on *PW* that states the effect of the process on the possible worlds. The process algebra also contains the underlying ADT as reduct. To make this more precise, we first rehearse some of the definitions relevant to equational specification of ADT's.

2.1. ADT specification

An *ADT signature Sig* is a pair $((S, \leq), \mathbb{F})$, where $(S, \leq)$ a poset of sort names and $\mathbb{F}$ a set of function declarations over S.

For each sort $s \in S$ of a signature we assume an infinite set X_s of *variables* of sort s. All X_s for all $s \in S$ are assumed to be mutually disjoint. So $X_{s_1} \cap X_{s_2} = \varnothing$ even if $s_1 \leq s_2$. The set of all variables is $X = \bigcup_{s \in S} X_s$. The set $T_{Sig,s}(X)$ of *Sig*-terms over X of sort s is defined in the usual way. The set of closed *Sig*-terms of sort s is $T_{Sig,s}$. We omit *Sig* is the signature is understood or irrelevant.

A *connected component* of S is an equivalence class with respect to the transitive symmetric closure of $\leq$. Two sort names are *compatible* if they are in the same connected component . A *Sig–equation* $t_1{=}t_2$ is a pair of terms over Sig_{ADT} whose least sorts of t_1 and t_2 are compatible. Goguen & Meseguer [12] show that under certain weak conditions, the terms of an ADT signature

always have a least sort.

Let Sig_{ADT} be an ADT signature. A Sig_{ADT}-specification is a pair (Sig_{ADT}, E), where E is a set of equations over Sig_{ADT}. We assume the initial semantics of ADT specifications [7, 10, 11, 12, 29].

2.2. Process specification with negation

By a "specification" we mean a set of axioms in a language that is used to select an intended model. By a "definition" we mean a formula that must be solved in a given model. The solution set of the formula in the model is the set of all elements of the model that satisfy the formula.

We extend the ADT specification $Spec_{ADT}$ to a process specification $Spec_{Process}$ by adding declarations of sort- and function names, and equations, where one sort in the process specification will be called the *process sort*. Terms of that sort are called *process terms*. The extension must be *conservative*, i.e. $Th(Spec_{ADT}) = Th(Spec_{Process}) \cap \mathcal{L}(Sig_{ADT})$, where $Th(Spec)$ is the set of theorems derivable from $Spec$ using equational logic, and $L(Sig)$ is the set of all possible equations over Sig. We do not require anything of the intended model of $Spec_{Process}$ except that it contains the initial algebra $\mathcal{A}_{ADT}$ of $Spec_{ADT}$ as a Sig_{ADT}-reduct, i.e. we will have

- $[\![s]\!]_{ADT} = [\![s]\!]_{Process}$ for $s \in S_{ADT}$, and
- $[\![f]\!]_{ADT} = [\![f]\!]_{Process}$ for $f \in \mathbb{F}_{ADT}$.

We now give our example $Spec_{Process}$.

```
process spec LibraryActions
    import
        Persons, Books, Library, Money
    sorts
        PERSON_EVENT, LIBRARY_EVENT, ACTION
    functions
        borrow  : BOOK           -> PERSON_EVENT
        return  : BOOK           -> PERSON_EVENT
        reserve : BOOK           -> PERSON_EVENT
        notify  : PERSON x BOOK  -> LIBRARY_EVENT
        pay     : MONEY          -> PERSON_EVENT
        _:_     : PERSON x PERSON_EVENT   -> ACTION
        _:_     : LIBRARY x LIBRARY_EVENT -> ACTION
end spec LibraryActions
```

`Person`, `Books`, and `Library` are ADT specifications. `Libary` declares a sort `LIBRARY` and a constant `l` of sort `LIBRARY`. (We consider a world with only one library.) The term `l:notify(p, b)` stands for the action of the library `l` notifying `p` that book `b` is overdue. **a**, **b**, and **c** are used as metavariables over closed terms of sort `ACTION`, ι and κ are metavariables over actors, and *a*, *b*, *c* are metavariables over events. We will often refer to *a* as an *atomic* event and to **a** as an *atomic* action.

By requiring the $Spec_{Process}$ to be a conservative extension of $Spec_{ADT}$, we know that `pay($2) = pay($1 + $1)`.

To be able to refer to persons and libraries as actors, we simply assume that the underlying ADT specification contains the specification

```
datatype spec LibraryActors
    import
        Persons, Library
    sorts
        PERSON < ACTOR
        LIBRARY < ACTOR
end spec LibraryActors
```

By the initial semantics, actors are either persons or libraries, but they are never books. By the reduct requirement, we know that each closed term of sort `ACTOR` denotes the same actor in each possible world. Each $t \in T_{\text{ACTOR}}$ can be used as an identifier of an object that rigidly designates the same object in all possible worlds. Assuming we have no equations for terms of sort `PERSON` or `LIBRARY`, it also uniquely denotes the same object in all possible worlds. The next specification specifies *steps*, processes that contain no sequence operator.

```
process spec StepAxioms
    import
        LibraryActions, LibraryActors
    sorts
        SCHEDULING_STEP
        ACTION < STEP
    functions
        any     : SCHEDULING_STEP
        fail    : SCHEDULING_STEP
        _+_     : STEP x STEP              -> SCHEDULING_STEP
        _&_     : STEP x STEP              -> SCHEDULING_STEP
        _:_     : ACTOR x SCHEDULING_STEP  -> STEP
        -_      : STEP                     -> STEP
    variables
        i, i0, i1, i2 : ACTOR
        s, s1, s2 : STEP
        e1, e2 : SCHEDULING_STEP
    equations
[S1]    i:(s1 + s2) = i:(s2 + s1)
[S2]    i:(s1 & s2) = i:(s2 & s1)
[S3]    --s = s
[S4]    if not ((i0 eq i1) and (i0 eq i2)) = true
        then -(i0:(i1:e1 + i2:e2)) = i0:(-i1:e1 & -i2:e2)
```

```
[S5]    if not ((i0 eq i1) and (i0 eq i2)) = true
        then -(i0:(i1:e1 & i2:e2)) = i0:(-i1:e1 + -i2:e2)
end spec StepAxioms
```

We use the convention that n-ary operators bind stronger than m-ary operators, $n<m$, but if we wish we can add brackets to emphasize operator binding. We use $\boldsymbol{\alpha}$ as metavariable over terms of sort STEP and α as metavariable over SCHEDULING_STEP. It is easily shown that $\boldsymbol{\alpha}$ either is a negated action or has the form $\iota{:}\alpha$. In general, $\iota{:}\alpha$ says that actor ι takes the initiative to the scheduling step α, or *schedules* α. α may be an atomic event a. In ι:**any**, ι executes any atomic event, and in ι:**fail**, ι deadlocks. $\iota{:}(\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2)$ denotes the choice of ι between $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$, $\iota{:}(\boldsymbol{\alpha}_1 \,\&\, \boldsymbol{\alpha}_2)$ denotes the choice of ι to let $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ occur synchronously, and $-\boldsymbol{\alpha}$ denotes the non-occurrence of $\boldsymbol{\alpha}$.

In [S4-5], we assume that a Boolean function eq with infix notation is defined for each actor sort, which is true iff its arguments are equal. The axioms require multi-actor steps to be a Boolean algebra. The intuition behind this is that the effect of not making a choice between α_1 and α_2 is the same as the effect of not doing α_1 and α_2. We call this the "extensional reading" of choice, by which we mean that they make De Morgan's laws valid in the semantics. In a single-actor step, we use what we will call an "intensional reading", and look at the choice of the single actor as an event in itself. On that reading, choice and synchronization are simply different actions and the axioms [S4-5] do not hold. We come back to this in the discussion after theorem 8, where the difference between the two choices is made more precise.

To make matters as simple as possible, we use the initial algebra of the process specification as intended semantics $\mathcal{A}_{Process}$ All closed process terms (and in particular all step terms) are then interpreted as their equivalence class with respect to the listed process and step axioms. This does not provide any meaning as far as the effect of a process on the set of possible worlds is concerned. The function ρ on $\mathcal{A}_{Process}$ will assign such a meaning, by defining $\rho(\beta)$ to be a (non-deterministic) function on possible worlds.

StepAxioms is notable for the axioms that are absent. For example, associativity of choice is standard in all process algebras. With initiative we would get

$$\iota{:}(\kappa{:}(\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2) + \boldsymbol{\alpha}_3) = \kappa{:}(\boldsymbol{\alpha}_1 + \iota{:}(\boldsymbol{\alpha}_2 + \boldsymbol{\alpha}_3))$$

and this is not a priori true intuitively, even if $\iota = \kappa$. ι and κ make completely different choices at the left-hand side and at the right-hand side. Moreover, ι and κ make their choices in a different order at the left- and right-hand sides. By omitting this axiom, we only have binary choices. Given the current axioms, an actor can only make a choice between n alternatives $n>2$ by making a sequence of binary choices, and these sequences are not equivalent.

Four other candidates for addition are

$\iota_0{:}(\iota_1{:}\alpha + \iota_1{:}\mathbf{any}) = \iota_1{:}\mathbf{any}$,
$\iota_0{:}(\iota_1{:}\alpha + \iota_1{:}\mathbf{fail}) = \iota_1{:}\alpha$,
$\iota_0{:}(\iota_1{:}\alpha \,\&\, \iota_1{:}\mathbf{any}) = \iota_1{:}\alpha$ and
$\iota_0{:}(\iota_1{:}\alpha \,\&\, \iota_1{:}\mathbf{fail}) = \iota_1{:}\mathbf{fail}$.

There are no derivability relations between these axioms, because for that we need the extra axioms

(6) $-\iota$:**any** = ι:**fail** and

(7) $-\iota$:**fail** = ι:**any**.

These last axioms enforce the intuitive interpretation "ι does something other than α, or deadlocks" to $-\iota:\alpha$. We will use the weaker interpretation "ι does something other than α, or deadlocks, or another actor does something", which is easier to formalize. A third, more stringent, interpretation would be "ι does something other than α, but does not deadlock", which is enforced by

(8) $\iota_0:(-\iota:\alpha + \iota:\alpha) = \iota$:**any** and

(9) $\iota_0:(-\iota:\alpha \,\&\, \iota:\beta) = \iota$:**fail**.

(6) and (7) are derivable from (8) and (9). We will keep our process specification simple by omitting all these axioms.

The sort PROCESS below is the process sort of our process specification.

```
process spec ProcessAxioms
    import
        StepAxioms
    sorts
        SCHEDULING_STEP < SCHEDULING_PROCESS
        STEP < PROCESS
    functions
        _+_ : PROCESS x PROCESS -> SCHEDULING_PROCESS
        _&_ : PROCESS x PROCESS -> SCHEDULING_PROCESS
        _:_ : ACTOR x SCHEDULING_PROCESS -> PROCESS
        _;_ : PROCESS x PROCESS          -> PROCESS
    variables
        i : ACTOR
        s : STEP
        p, p1, p2, p3 : PROCESS
    equations
[P1]    (p1 ; p2) ; p3 = p1 ; (p2 ; p3)
[P2]    i:(p1 + p2) = i:(p2 + p1)
[P3]    i:(p1;p3 + p2;p3) = i:(p1 + p2) ; p3
[P4]    i:fail ; i:s = i:fail
[P5]    i:fail ; p ; i:s = i:fail ; p
end spec ProcessAxioms
```

To keep matters simple, we do not allow negation of processes, although this has been formalized in an earlier version of the language [6, 24] and can easily be added here. We use β as metavariable over terms of sort PROCESS. We use the convention that : binds stronger than ;. Note that if i deadlocks, other actors can still display initiative as if nothing happened.

We are able to express three different kinds of nondeterminism in this specification. Choice is "non-deterministic" in the sense that it is *arbitrary*; it is not specified which branch is taken. In a

two-actor environment with actors M and O, $M:(\alpha_1 + \alpha_2)$ is "non-deterministic" in the sense that O has no *control* over the choice (this is CSP-like nondeterminism [15]). Finally, $\iota_0:(\iota:a\ ;\ x + \iota:a\ ;\ y)$ is the usual concept of nondeterminism as formalized in process algebra [2, 3, 4], where there action $\iota:a$ can lead to one out of a set of possible states.

2.3. Dynamic logic with equality

Dynamic logic includes first-order predicate logic, which we define first. A *static constraint signature* *Sig* is a triple $((S,\ \leq), \mathbb{F}, \mathbb{P})$, where $((S,\ \leq), \mathbb{F})$ is an ADT signature and $\mathbb{P}$ is a set of *predicate symbols* with their arity taken from S.

An *atomic Sig-formula* over a static constraint signature is either a *Sig*-equation or a formula $P(t_1, \ldots, t_n)$, where P is declared to have arity $s_1 \times \ldots \times s_n$ and $t_i \in T_{Sig,\ s_i}(X)$, $i = 1, \ldots, n$. We use P, Q as metavariables over the predicate symbols. An *order-sorted static constraint language* $L_{Stat}(Sig)$ over a static constraint signature Sig is the set of formulas defined by the following BNF:

$$\phi ::= t_1 = t_2 \mid P(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x(\phi) \mid \exists x(\phi),$$

where the first two alternatives are atomic *Sig*-formulas. We drop the qualification *Sig* if it is clear from the context, or if it irrelevant which static constraint signature is used. Punctuation symbols (,) are used to disambiguate formulas. Equations are treated as universally quantified formulas in which the binary =-predicate appears in infix notation. This makes the standard rules for substitution and variable binding applicable to equations that appear in formulas. The set of all formulas over Sig_{Stat} is called $L(Sig_{Stat})$. A *static constraint specification* is a pair Sig_{Stat}, C_{Stat}), where C_{Stat} is a set of closed formulas over Sig_{Stat}.

A *dynamic constraint signature* Sig_{Dyn} consists of a triple of signatures (Sig_{ADT}, Sig_{Stat}, $Sig_{Process}$) such that Sig_{Stat} and $Sig_{Process}$ are static IC and process signatures, respectively, that share Sig_{ADT} as their underlying ADT specification. Other than that, $Spec_{Stat}$ and $Spec_{Process}$ have no sort - or operation names in common.

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

$$\Phi ::= \phi \mid \Phi_1 \vee \Phi_2 \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \rightarrow \Phi_2 \mid \Phi_1 \leftrightarrow \Phi_2 \mid [\beta]\Phi$$

where ϕ is a formula of $L_{Stat}(Sig_{Stat})$ and β is a process term over $Sig_{Process}$. We use

$$\langle\beta\rangle\Phi$$

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

The intuitive semantics of $[\beta]\Phi$ is "after execution of β, Φ holds necessarily", and the intuitive semantics of $\langle\beta\rangle\Phi$ is "after execution of β, Φ may hold". The language can thus be used to express pre- and postconditions of events. Note that by our definition of $L_{Dyn}(Sig_{Dyn})$-formulas, process terms can only occur inside the modal operator. A dynamic logic formula containing no modal operators is a static constraint formula, which may be an equation over Sig_{ADT}.

$Spec_{Dyn} = (Sig_{Dyn}, C_{Dyn})$ is a *dynamic constraint specification* over the static constraint specification $Spec_{Stat}$ and the process specification $Spec_{Process}$ if it is an extension of $Spec_{Stat}$ and $Spec_{Process}$ such that

$Th(Spec_{Stat}) = Th(Spec_{Dyn}) \cap L_{Stat}(Sig_{Stat})$ and
$Th(Spec_{Process}) = Th(Spec_{Dyn}) \cap L_{Process}(Sig_{Process})$.

A dynamic constraint specification thus conservatively extends both the static constraint specification and the process specification. An example dynamic constraint specification is

```
dynamic constraint spec DynamicLibrary
    import
        StaticLibraryConstraints, ProcessAxioms
    variables
        p : PERSON
        b : BOOK
        q : QUEUE
    dynamic constraints
[D0]    [p:borrow(b)] Borrowed(b, p) and not Present(b)
[D1]    if Reserved(b) then
          (if p = head(q) and q = reservations(b)
           then [p:borrow(b)] reservations(b) = tail(q)
end spec DynamicLibrary
```

[D0] says that the effect of borrowing a book is that it is not present and is borrowed. [D1] says that in the case of reserved books, we remove the borrower from the queue of reservations if he is the first in the queue, otherwise the event is blocked.

2.4. Inference rules and semantics

Let $Spec_{Dyn} = (Sig_{Dyn}, C_{Dyn})$ be a dynamic constraint specification over $Spec_{Stat}$ and $Spec_{Process}$. A model of $Spec_{Dyn}$ is a four-tuple

$$\mathcal{M}_{Dyn} = (\mathcal{A}_{ADT}, PW, \mathcal{A}_{Process}, \rho)$$

such that

- $\mathcal{A}_{ADT}$ is a model of $Spec_{ADT}$,
- PW is a model of $Spec_{Stat}$,
- $\mathcal{A}_{Process}$ is a model of $Spec_{Process}$,
- $\rho: T_{Process}(X) \longrightarrow (\Sigma \longrightarrow (PW \longrightarrow \mathcal{P}(PW)))$, where X is the set of all variables defined in section 2.1, $T_{Process}(X)$ is the set of process terms of $Sig_{Process}$, and Σ is the set of all sort-preserving assignments σ to variables,
- ρ satisfies the requirement that for $\beta_1, \beta_2 \in T_{Process}(X)$,

$$[\![\beta_1]\!]_{\sigma,\mathcal{A}_{Process}} = [\![\beta_2]\!]_{\sigma,\mathcal{A}_{Process}} \text{ implies } \rho(\beta_1)(\sigma) = \rho(\beta_2)(\sigma).$$

All sort names have a fixed interpretation in all possible worlds, and if $x: s$ and σ is a sort-preserving assignment, then $\sigma(x)$ is an element of $[\![s]\!]_{\mathcal{A}}$, where $\mathcal{A}$ is either $\mathcal{A}_{ADT}$ or $\mathcal{A}_{Process}$. The denotation function $[\![.]\!]$ does double duty as semantics of sort and function names, as well as of terms. So we have

$[\![\cdot]\!]_{\sigma,\mathcal{A}} : T_s(X) \longrightarrow [\![s]\!]_{\mathcal{A}}$ for a sort s.

ρ is called the *behavioral* semantics of process terms, the denotation $[\![\beta]\!]_{\mathcal{A}_{Process}}$ is called the *equational* semantics of process term. The requirement on ρ is called the *congruence requirement on the behavioral semantics*. It guarantees that processes that are equal in the process algebra, have equal effects on the world.

If $\mathcal{M}_{Dyn} = (\mathcal{A}_{ADT}, PW, \mathcal{A}_{Process}, \rho)$ is a model of Sig_{Dyn} and $\sigma: X \longrightarrow \mathcal{M}_{Dyn}$ ia an assignment to all variables, then truth in $w \in PW$ under assignment σ of a dynamic logic formula is defined by:

- for each $\beta \in T_{Process}(X)$, we have $w, \sigma \models_{Dyn} [\beta]\Phi$ iff for all $w' \in \rho(\beta)(\sigma)(w)$, we have $w', \sigma \models_{Dyn} \Phi$.
- Truth of the other dynamic logic formulas is defined as usual.

For each σ, we define $\mathcal{M}_{Dyn}, \sigma \models_{Dyn} \Phi$ iff $w, \sigma \models_{Dyn} \Phi$ for all $w \in PW$.

Truth of a formula in a world ($w \models_{Dyn} \Phi$), in a model ($\mathcal{M}_{Dyn} \models_{Dyn} \Phi$), and of a specification in a model ($\mathcal{M}_{Dyn} \models_{Dyn} Spec_{Dyn}$) are defined as usual.

This truth definition coincides with the standard truth definition for ϕ in $L(Sig_{Stat})$, which we denote $\models_{Stat}$, and with the standard truth definitions for equations in $L(Sig_{ADT})$, which we denote $\models_{ADT}$. We have for all $w \in PW$ that

$w \models_{Stat} \phi$ iff $w \models_{Dyn} \phi$ and
$w \models_{ADT} t_1 = t_2$ iff $w \models_{Stat} t_1 = t_2$.

We will therefore omit the subscript from $\models$ from now on.

The inference rules of dynamic logic are given in table 1.

[MP] $\dfrac{\Phi,\ \Phi \rightarrow \Psi}{\Psi}$	[G] $\dfrac{\Phi}{\forall x(\Phi)}$	[N] $\dfrac{\Phi}{[\beta]\Phi}$
[Ref] $t = t$	[Sym] $t_1 = t_2 \rightarrow t_2 = t_1$	[Tran] $(t_1 = t_2 \wedge t_2 = t_3) \rightarrow t_1 = t_3$
[Con1] $\dfrac{t_1 = t_2}{t\{x \mapsto t_1\} = t\{x \mapsto t_2\}}$		[Sub1] $\dfrac{t_1 = t_2}{t_1\{x \mapsto t\} = t_2\{x \mapsto t\}}$
[Con2] $\dfrac{t_1 = t_2}{P(t_1) \leftrightarrow P(t_2)}$		[Sub2] $\dfrac{\beta_1 = \beta_2}{([\beta\{x \mapsto \beta_1\}]\Phi \leftrightarrow [\beta\{x \mapsto \beta_2\}]\Phi)}$
All axioms and theorems of first-order logic.		
[DL1] $[\beta](\Phi_1 \rightarrow \Phi_2) \rightarrow ([\beta]\Phi_1 \rightarrow [\beta]\Phi_2)$		

Table 1. Inference rules for dynamic logic with equality

The equality axioms [Ref], [Sym] and [Tran] hold for all terms, including process terms. If Φ is derivable from a set H of L_{Dyn} formulas, we write $H \vdash_{Dyn} \Phi$.

It must be noted which axioms from standard dynamic logic are absent, viz.

[DL2] $[\beta_1;\beta_2]\Phi \leftrightarrow [\beta_1]([\beta_2]\Phi)$

[DL3] $[\iota:(\beta_1 + \beta_2)]\Phi \leftrightarrow [\beta_1]\Phi \wedge [\beta_2]\Phi$,

[DL4] $[\iota:(\alpha_1 \,\&\, \alpha_2)]\Phi \leftarrow [\alpha_1]\Phi \vee [\alpha_2]\Phi$,

[DL5] $[\iota:\mathbf{fail}]\Phi$.

It is again easy to see that these inference rules monotonically extend the corresponding rules $\vdash_{Stat}$ and $\vdash_{ADT}$ for $L(Sig_{Stat})$ and $L(Sig_{ADT})$. We have

$Spec_{Dyn} \vdash_{Stat} \phi$ iff $Spec_{Dyn} \vdash_{Dyn} \phi$ and
$Spec_{Dyn} \vdash_{ADT} t_1 = t_2$ iff $Spec_{Dyn} \vdash_{Dyn} t_1 = t_2$.

We drop the subscript from $\vdash$ from now on.

Theorem 1.

$Spec_{Dyn} \vdash \Phi$ iff $Spec_{Dyn} \models \Phi$.

Sketch of proof.

Soundness is easy to prove. As a sketch of completeness, first note that [N] and [DL1] characterize Kripke models, with accesibility relations R_β for each $\beta \in T_{Process}(X)$. Moreover, the derived inference rule $\dfrac{\beta_1 = \beta_2}{[\beta_1]\Phi \leftrightarrow [\beta_2]\Phi}$ corresponds to the property $R_{\beta_1} = R_{\beta_2}$ for $\beta_1 = \beta_2$ on Kripke models of this kind. By standard modal techniques [17], this implies that every consistent formula can be satisfied by a Kripke model with this property. By contraposition, we obtain completeness under the assumption that we can always prove $\beta_1 = \beta_2$ whenever this holds. But the truth of that assumption follows from the completeness of equational logic. ■

Theorem 2.

The following theorems hold in $Spec_{Dyn}$.

[Dyn1] $[\beta]\mathbf{true}$

[Dyn2] $\langle\beta\rangle\mathbf{false}$

[Dyn3] $[\beta](\Phi_1 \wedge \Phi_2) \leftrightarrow ([\beta]\Phi_1 \wedge [\beta]\Phi_2)$

[Dyn4] $[\beta](\Phi_1 \vee \Phi_2) \leftarrow ([\beta]\Phi_1 \vee [\beta]\Phi_2)$

[Dyn5] $\langle\beta\rangle(\Phi_1 \vee \Phi_2) \leftrightarrow (\langle\beta\rangle\Phi_1 \vee \langle\beta\rangle\Phi_2)$

[Dyn6] $[-\iota:(\alpha_1 + \alpha_2)]\Phi \leftrightarrow [\iota:(-\alpha_1 \,\&\, -\alpha_2)]\Phi$

[Dyn7] $[-\iota:(\alpha_1 \,\&\, \alpha_2)]\Phi \leftrightarrow [\iota:(-\alpha_1 + -\alpha_2)]\Phi$

[Dyn8] $[-(-\alpha)]\Phi \leftrightarrow [\alpha]\Phi$

Proof.

For [Dyn1-5], see [24]. [Dyn6-8] follow from [S2] and the step axioms for our process theory. ■

These theorems do not relate the internal structure of Φ to the internal structure of β in $[\beta]\Phi$. The truth of formulas that express such a relation depends upon the behavioral semantics of process terms, to which we now turn.

2.5. A model for free choice

Assuming the initial semantics for $Spec_{ADT}$ and $Spec_{Process}$, all the models of a dynamic specification differ only in their behavioral semantics, i.e. in the function ρ. We therefore select a particular intended model by defining the function ρ. First, we assume a function

$$effect: T_{\mathrm{EVENT}}(X) \longrightarrow (\Sigma \longrightarrow (PW \longrightarrow PW)).$$

$effect(\mathbf{a})(\sigma)$ defines the effect of an atomic event on the state of the world. In general, a dynamic logic specification does not determine the effect of events exhaustively. Several *effect* functions remain possible with respect to a given specification, and we need a kind of frame assumption to choose between these possibilities. For example, one can stipulate that whatever is not specified to change does not change when an atomic event is applied. We leave open how *effect* is chosen, and require only that the function satisfies

(E) if $[\![\mathbf{a}]\!]_\sigma = [\![\mathbf{b}]\!]_\sigma$ then $effect(\mathbf{a})(\sigma) = effect(\mathbf{b})(\sigma)$,

where the model $\mathcal{M}_{Dyn}$ is left understood. To define

$$\rho: T_{\mathrm{STEP}}(X) \longrightarrow (\Sigma \longrightarrow (PW \longrightarrow \mathcal{P}(PW))),$$

we define a set of functions $PW \longrightarrow PW$ to each $\alpha \in T_{\mathrm{STEP}}(X)$. We do this inductively. First, we need the domain of *synchronization sets*

Definition 3.

1. An element of $\mathcal{F}^+(T_{\mathrm{EVENT}}(X) \cup \{\iota\mathbf{:fail}\}_{\iota \in T_{\mathrm{ACTOR}}})$ is called a *synchronization set*. ($\mathcal{F}^+$ is the finite non-empty subset operator.) Synchronization sets are written $\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ . \\ . \\ . \end{bmatrix}$.

2. A synchronization set $\begin{bmatrix} \mathbf{a}_1 \\ . \\ . \\ . \\ \mathbf{a}_n \end{bmatrix}$ not containing failure events is called *compatible* if for all $w \in PW$ and all $\sigma \in \Sigma$

$$effect(\mathbf{a}_1)(\sigma) \circ \ldots \circ effect(\mathbf{a}_n)(\sigma)(w) = effect(\mathbf{a}_{i_1}) \circ \ldots \circ effect(\mathbf{a}_{i_n})(w)$$

for all permutations $\langle i_1, \ldots, i_n \rangle$ of $\langle 1, \ldots, n \rangle$. A synchronization set $s_1 \cup s_2$ where s_1 contains no failure events and s_2 contains only failure events is compatible if s_1 is compatible and there is no actor ι such that $\iota{:}a \in s_1$ and $\iota\mathbf{:fail} \in s_2$ for an a.

3. The set of all (compatible and incompatible) synchronization sets is called *SYN* and has metavariable s. The set of ι-synchronization events is $SYN^\iota = \{s \in SYN \mid \iota{:}a \in s \text{ for an } a\}$. ■

Definition 4.

If $s \in SYN$ is compatible, then we define $effect(s) \in (\Sigma \longrightarrow (PW \longrightarrow \mathcal{P}(PW)))$ by

$$effect(s)(\sigma) = effect(\mathbf{a}_1)(\sigma) \circ \ldots \circ effect(\mathbf{a}_n)(\sigma)$$

with $s = \begin{bmatrix} \mathbf{a}_1 \\ . \\ . \\ . \\ \mathbf{a}_n \end{bmatrix} \cup s^f$, where s^f contains only elements of the form ι:**fail** and none of the $\mathbf{a}_i$ has this form.

■

In our step specification, we have

$$\texttt{ACTION} < \texttt{STEP},$$

where `ACTION` is the sort of terms $\iota:a$, for a an elementary event. `STEP` is built from terms of the form $\iota:\alpha$, where α is a scheduling step, i.e. a term containing `any`, `fail`, `-`, `+`, or `;`.

Definition 5.

The set of steps is defined as

$$STEP = \mathcal{P}^+(\mathcal{P}(SYN)).$$

The function

$$step: T_{\texttt{STEP}}(X) \cup \{\iota:\mathbf{any}, \iota:\mathbf{fail} \mid \iota \in T_{\texttt{ACTOR}}\} \longrightarrow STEP$$

is defined inductively as follows.

1. For $T_{\texttt{ACTION}}(X)$-terms, we define $step(\mathbf{a}) = \{s \in SYN \mid \mathbf{a} \in s$ and s compatible$\}$ for $\mathbf{a} \in T_{\texttt{ACTION}}(X)$.
2. $step(\iota:\mathbf{fail}) = \{s \in SYN \mid \iota:\mathbf{fail} \in s$ and s is compatible$\}$.
3. $step(\iota:\mathbf{any}) = \{s \in SYN^\iota \mid s$ is compatible$\}$. ■

A step is thus a set of the form

$$\{\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ . \\ . \\ . \end{bmatrix}, \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ . \\ . \\ . \end{bmatrix}, \ldots\}.$$

$step(\mathbf{a})$ is the step consisting of all synchronization sets in which $\mathbf{a}$ participates. It is an element of $\mathcal{P}(SYN)$, so the codomain of $step$ is $\mathcal{P}(\mathcal{P}(SYN))$. Because $step$ never maps a step term to the empty set, we can eliminate this from the codomain.

The behavioral interpretation of $\mathbf{a}$ thus involves nondeterminism of the underspecification variety, because it is interpreted as "$\mathbf{a}$ occurs, together with a finite set of other events, but I don't know (specify, care) which other events". This interpretation is crucial for the possibility to give a behavioral interpretation to action negation. We first give a behavioral semantics for step terms only, and then make a minor modification, which gives us the intended behavioral semantics for all process terms.

Definition 6.

We define the free choice model $\mathcal{M}_f$ as follows. Define

$$\rho: T_{\text{STEP}}(X) \longrightarrow (\Sigma \longrightarrow (PW \longrightarrow \mathcal{P}(PW)))$$

inductively by

1. $\rho(\mathbf{a})(\sigma)(w) = \{effect(s)(\sigma)(w) \mid \text{for all } s \in step(\mathbf{a})\}$.
2. $\rho(\iota{:}\mathbf{fail})(\sigma)(w) =$ as in 1, with **a** replaced by ι:**fail**.
3. $\rho(\iota{:}\mathbf{any})(\sigma)(w) =$ as in 1, with **a** replaced by ι:**any**.
4. $\rho(-\boldsymbol{\alpha})(\sigma)(w) = \{effect(s)(\sigma)(w) \mid s \in SYN \backslash step(\boldsymbol{\alpha})\}$.
5. $\rho(\iota_0{:}(\iota_1{:}\alpha_1 + \iota_2{:}\alpha_2))(\sigma)(w) = \rho(\iota_1{:}\alpha_1)(\sigma)(w) \cup \rho(\iota_2{:}\alpha_2)(\sigma)(w)$, where ι_0, ι_1, and ι_2 are not all equal.
6. $\rho(\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2))(\sigma)(w) = \rho(\iota{:}\alpha_1)(\sigma)(w) \cap \rho(\iota{:}\alpha_2)(\sigma)(w)$.
7. $\rho(\iota_0{:}(\iota_1{:}\alpha_1 \;\&\; \iota_2{:}\alpha_2))(\sigma)(w) = \rho(\iota_1{:}\alpha_1)(\sigma)(w) \cap \rho(\iota_2{:}\alpha_2)(\sigma)(w)$. ■

Remarks:

1. *step* (**a**)(σ) is the set of all compatible synchronization sets in which **a** participates, so the order of applying the *effect* function in 1 is immaterial.
2. $\rho(\iota{:}\mathbf{fail})(\sigma)(w)$ is the set of all compatible synchronization sets containing ι:**fail**. By the definition of compatibility, it will not contain any $\iota{:}a$.
3. $\rho(\iota{:}\mathbf{any})(\sigma)(w)$ contains all compatible synchronization sets in which ι participates, but not with a ι:**fail**.
4. Actions are negated with respect to all possible steps. Thus, $-\iota{:}a$ is the set of all possible synchronization sets in which ι does not participate with a. These are the synchronization sets in which ι participates with another event, or with **fail**, or in which ι does not participate at all.
5. Choice makes ρ deliver a function $PW \longrightarrow \mathcal{P}(PW)$ rather than $PW \longrightarrow PW$. In general, each set has a *set* of possible next worlds, one of which will be actually reached when executing the step. The intuitive notion captured by this semantic definition is that if ι_0 chooses between two steps, the set of possible next world that may result from his choice is the union of the sets of possible next worlds reachable by the branches. Note that, if $\rho(\boldsymbol{\alpha}_i)(\sigma)(w)$ for $i = 1, 2$ consists of compatible synchronization sets, then so does $\rho(\iota{:}(\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2))(\sigma)(w)$.
6. If the three actors involved are the same, then we capture a quite different intuition with the semantics, viz. what the effect of ι's choice itself is, rather than what the effect of the chosen branches is. It says that the effect of the making the choice itself is the intersection of the effects of the branches. This agrees with our stronger interpretation of $P(\iota{:}(\iota{:}a + \iota{:}b))$, which says that ι is permitted to do a and b (just as in the multi-actor case), and in addition that ι is permitted to make the choice.
7. There are no special considerations for synchronization that distinguish the single-actor from the multi-actor case. $\boldsymbol{\alpha}_1 \,\&\, \boldsymbol{\alpha}_2$ is the synchronous execution of two steps, and the effect will be brought about by those functions on PW that bring about the effect of $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$. Hence, we intersect the steps. Note that the effect of this synchronous execution is the same as the effect of the choice event in a single-actor choice between $\iota{:}\alpha_1$ and $\iota{:}\alpha_2$. This is compatible with the process terms

whose effect this is, being unequal. Note that α_1 and α_2 may be *incompatible*, so that there is no synchronization set in which both participate. For incompatible steps α_1 and α_2, we have $\rho(\iota{:}(\alpha_1 \,\&\, \alpha_2))(\sigma)(w) = \varnothing$. This has an important consequence for the logic, which we will see below.

We call this model a *free choice* model because it allows us to differentiate a choice imposed on an actor by another actor from a choice made freely by the actor himself.

Theorem 7.

$\mathcal{M}_f$ is a model of $Spec_{Dyn}$.

Sketch of proof.

We simply prove that ρ applied to both sides of each axiom in `StepAxioms` yields the same result. This equality will then be maintained in any equation derivable from the axioms. The proof is simple and is omitted. ■

Theorem 8.

If ι_0, ι_1 and ι_2 are not all equal, then the following formulas are true in $\mathcal{M}_f$.

[F1] $[\iota_0{:}(\iota_1{:}\alpha_1 + \iota_2{:}\alpha_2)]\Phi \leftrightarrow [\iota_1{:}\alpha_1]\Phi \wedge [\iota_2{:}\alpha_2]\Phi$

[F2] $\langle\iota_0{:}(\iota_1{:}\alpha_1 + \iota_2{:}\alpha_2)\rangle\Phi \leftrightarrow \langle\iota_1{:}\alpha_1\rangle\Phi \vee \langle\iota_2{:}\alpha_2\rangle\Phi$

[F3] $[\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2)]\Phi \leftarrow [\iota{:}\alpha_1]\Phi \vee [\iota{:}\alpha_2]\Phi$

[F4] $\langle\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2)\rangle\Phi \rightarrow \langle\iota{:}\alpha_1\rangle\Phi \wedge \langle\iota{:}\alpha_2\rangle\Phi$

[F5] $[\iota_0{:}(\iota_1{:}\alpha_1 \,\&\, \iota_2{:}\alpha_2)]\Phi \leftarrow [\iota_1{:}\alpha_1]\Phi \vee [\iota_2{:}\alpha_2]\Phi$

[F6] $\langle\iota_0{:}(\iota_1{:}\alpha_1 \,\&\, \iota_2{:}\alpha_2)\rangle\Phi \rightarrow \langle\iota_0{:}\alpha_1\rangle\Phi \wedge \langle\iota_1{:}\alpha_2\rangle\Phi$

■

Remarks:

1. [F2] represents our "extensional" reading of a multi-actor choice. Φ may be true after ι_0's choice iff it may be true as an effect of α_1 or α_2. By the duality $[\alpha]\Phi \equiv \neg\langle\alpha\rangle\neg\Phi$, we have [F1], saying that Φ will be true after ι_0's choice iff it will be true after α_1 and α_2.
2. In a single-actor choice, [F4] reflects our "intensional" reading that if Φ may be the effect of ι's choice, then it may be the effect of each of the branches. The arrow goes only one way, because there may not be any joint effect of α_1 and α_2 at all, viz. when they are incompatible. However, if there is an effect, i.e. if there is a Φ with $\langle\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2)\rangle\Phi$, then we can conclude that $\langle\iota{:}\alpha_1\rangle\Phi \wedge \langle\iota{:}\alpha_2\rangle\Phi$. To understand the dual formula in [F3], we must realize that choice is an event that occurs before the branches are executed. Choice does not bring us to a next possible world, but it does occur at a point in time preceding the execution of α_1 and α_2. The left-hand side of [F4] then says

   "after ι's choice, the system is in a state where Φ can be brought about",

   which implies the right-hand side, both branches can bring about Φ. Applying the duality, the left-hand side becomes

   "it is not the case that after ι's choice, the system is in a state where $\neg\Phi$ can be brought about",

which is equivalent to

"after ι's choice, the system can be in a state where Φ will be brought about".

This is the correct reading of the left-hand side of [F3], and it is implied by the right-hand side, which says that one of the branches will bring about Φ.

3. Synchronous execution is impossible if the synchronized steps are incompatible, so there is a one-way arrow here as well. The logic is not able to express necessary conditions for two steps to be compatible. This is a general problem with the intersection of accessibility relations in a Kripke model with multiple accessibility relations, that can only be solved by strengthening the language. Meyer [25] did this by adding *DONE*:α predicates to the language, but Van der Hoek and Meyer [16] show how to do this in general.

To give a behavioral semantics of process terms, we must extend the definition of ρ to $T_{\mathtt{PROCESS}}(X)$. The basic idea is simply that the effect of the sequence operator ; on the world is simply a composition of the effect functions of its arguments. The only complication is that deadlock should remain local, so that, for example (see `[P4-5]` in `ProcessAxioms`)

$$\rho(\iota{:}\mathbf{fail}\ ;\ \beta_1\ ;\ \iota{:}\alpha\ ;\ \beta_2) = \rho(\iota{:}\mathbf{fail}\ ;\ \beta_1\ ;\ \beta_2).$$

Other actors can continue even when ι gets stuck. The easiest way is to define ρ for terms in which all occurrences of ι after it has failed are removed, and then extend the definition to other process terms that are in its congruence class.

Definition 10.

Let $\beta \in T_{\mathtt{PROCESS}}(X)$.

1. Every nested application of the ; operator in β is called a *sequence*.
2. A sequence is called *redundant* if it contains a pattern $\iota{:}\mathbf{fail}\ ;\ \Box\ ;\ \iota{:}\theta\ ;\ \Box$, where θ is either α or **fail**, and □ is a (possibly empty) context.
3. A sequence is called *non-redundant* if it is not redundant.
4. β is called non-redundant if it does not contain redundant sequences. ■

The following is easy to prove.

Lemma 11.

In the specification `ProcessAxioms`, For any $\beta \in T_{\mathtt{PROCESS}}(X)$ there is a unique non-redundant `PROCESS`-term equal to it. ■

Definition 12.

Extend the definition of ρ as follows.

$$\rho\colon T_{\mathtt{PROCESS}}(X) \longrightarrow (\Sigma \longrightarrow (PW \longrightarrow \mathcal{P}(PW)))$$

is defined inductively by

1. If $\beta \in T_{\mathtt{STEP}}(X)$, then use the definition for `STEP`-terms.
2. If β is of the form $\iota{:}(\beta_1 + \beta_2)$ or $\iota{:}(\beta_1\ \&\ \beta_2)$, then use the corresponding definition for `STEP`-terms.

3. Otherwise, let $\hat{\beta}$ be the unique non-redundant process term equal to β, and let $\hat{\beta} = \beta_1 ; \beta_2$ (any arbitrary decomposition will do). Then by the inductive build-up, $\rho(\beta_i)(\sigma)$ is a function $PW \longrightarrow \mathcal{P}(PW)$, for $i = 1, 2$. Then define $\rho(\beta)(\sigma)(w) = \{f_2 \circ f_1 \mid f_i \in \rho(\beta_i)(\sigma), i = 1, 2\}$, where $\circ$ is the usual function composition lifted to sets. ■

Theorem 13.

$\mathcal{M}_f$ is a model of $Spec_{Dyn}$. ■

Theorem 14.

The following two process logic axioms are true in $\mathcal{M}_f$.

[F7] $[\beta_1;\beta_2]\Phi \leftrightarrow [\beta_1]([\beta_2]\Phi)$

[F8] $\langle\beta_1;\beta_2\rangle\Phi \leftrightarrow \langle\beta_1\rangle(\langle\beta_2\rangle\Phi)$ ■

3. Deontic logic

$L(Sig_{Dyn})$ can be used as a language for deontic constraints by introducing *violation states* V_i: $\iota{:}\alpha$, one for each of the reasons why ι would be forbidden to perform α. Then deontic modalities are introduced as follows.

Definition 15.

The following abbreviations are used for deontic modalities:

- $\mathbf{P}(\alpha) \overset{\Delta}{\Leftrightarrow} \neg[\alpha]V_i{:}\alpha$ for an i, (" α is permitted"),
- $\mathbf{O}(\alpha) \overset{\Delta}{\Leftrightarrow} [-\alpha]V_i{:}\alpha$ for an i (" α is obligatory"),
- $\mathbf{F}(\alpha) \overset{\Delta}{\Leftrightarrow} \neg\, \mathbf{P}(\alpha)$ ("α is forbidden"). ■

This formalization of deontic logic has been studied in dynamic logic without actors in [24, 25] and has been applied to system specification in [31, 30]. With actors, the modalities express more. For example, $\mathbf{P}(\iota{:}a)$ says that ι is permitted do to a, and $\mathbf{P}(\iota{:}(\iota{:}a + \iota{:}b))$ says that ι is permitted to choose between doing a or b (i.e. choosing brings him into a state where he can do a permitted action).

Theorem 16.

If ι_0, ι_1 and ι_2 are not all equal, then the following formulas are true in $\mathcal{M}_f$.

[P1] $\mathbf{F}(\iota_0{:}(\iota_1{:}\alpha_1 + \iota_2{:}\alpha_2)) \leftrightarrow \mathbf{F}(\iota_1{:}\alpha_1) \wedge \mathbf{F}(\iota_2{:}\alpha_2$

[P2] $\mathbf{P}(\iota_0{:}(\iota_1{:}\alpha_1 + \iota_2{:}\alpha_2)) \leftrightarrow \mathbf{P}(\iota_1{:}\alpha_1) \vee \mathbf{P}(\iota_2{:}\alpha_2)$

[P3] $\mathbf{F}(\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2)) \leftarrow \mathbf{F}(\iota{:}\alpha_1) \vee \mathbf{F}(\iota{:}\alpha_2)$

[P4] $\mathbf{P}(\iota{:}(\iota{:}\alpha_1 + \iota{:}\alpha_2)) \rightarrow \mathbf{P}(\iota{:}\alpha_1) \wedge \mathbf{P}(\iota{:}\alpha_2)$

[P5] $\mathbf{F}(\iota_0{:}(\iota_1{:}\alpha_1 \;\&\; \iota_2{:}\alpha_2)) \leftarrow \mathbf{F}(\iota_1{:}\alpha_1) \vee \mathbf{F}(\iota_2{:}\alpha_2)$

[P6] $\mathbf{P}(\iota_0{:}(\iota_1{:}\alpha_1 \;\&\; \iota_2{:}\alpha_2)) \rightarrow \mathbf{P}(\iota_0{:}\alpha_1) \wedge \mathbf{P}(\iota_1{:}\alpha_2)$

■

Remarks:

1. [P2] says that ι_0 is permitted to give two actors a choice iff at least one of the actors can do something permitted. ι_0 is permitted to choose, because his choice may lead to a permitted state of the world. By duality, he is forbidden to do so iff both actors are given forbidden things to do ([P1]).
2. [P4] says that if ι is permitted to make a choice between α_1 and α_2, then he is permitted to perform either branch. By duality, if he is forbidden to do either branch, then he is forbidden to make the choice ([P3]). This resolves the paradox if free choice permission.
3. If any actor is permitted to synchronize two steps, then both steps are permitted([P6]). If at least one step is forbidden, then any actor is forbidden to synchronize them ([P5]).

4. Discussion

De Nicola and Hennessy [27] give a translation function **tr** from CCS to TCCS which deletes all occurrences of τ and replaces choice by internal or external choice, depending upon the first event of the chosen branches. This translation makes sense if we restrict ourselves to a two-actor system, where one actor (the observer o) initiates all atomic events and either actor (o or the machine m) can initiate a choice. Example translations (in our actor formalism) are

1. $\mathbf{tr}(a + b) = o:(o:a + o:b)$
2. $\mathbf{tr}(\tau a + b) = m:(o:(o:a + o:b) + o:a)$
3. $\mathbf{tr}(\tau a + \tau b) = m:(o:(o:a + o:b) + o:a + o:b)$.

The last translation assumes choice is associative, which is not obvious in an actor-oriented specification. Making actors explicit shows that there are more ways to interpret τ. For example, the translation in 2 says that m chooses between letting o do a or giving o the choice to do a or b. A translation that stays closer to the original would be

2'. $\mathbf{tr}(\tau a + b) = o:((m:c\ ;\ o:a) + o:b)$

for an event c, and assuming CCS choice is external. One can argue that o is not able to make the choice in the right-hand side of 2' because o doesn't know what m will do, but then neither is m in the right-hand side of 2 able to make a choice. Perhaps the problem in 2 is that the initiative of the choice lies partly with m and partly with o (if o is fast enough, he can press the button before m does τ).

This problem is also present in 3, where the intention is that the choice is made by m. A simpler translation would therefore be

3'. $\mathbf{tr}(\tau a + \tau b) = m:(o:a + o:b)$.

This is certainly not the same thing as

3''. $\mathbf{tr}(\tau a + \tau b) = o:(m:c_1\ ;\ o:a + m:c_2\ ;\ o:b)$,

which is another reading of $\tau a + \tau b$. We do not argue for either of these translations in favor of the other, but want to point out that using actors, one is forced to make explicit which choice one makes.

The problem that an actor cannot choose if he has not enough information is illustrated neatly by a number of CSP laws [15, pages 103-107]. Translated into process terms with actors, these are:

1. $m:(o:a\ ;\ x + o:a\ ;\ y) = o:a\ ;\ m:(x + y)$
2. $o:(o:a\ ;\ x + o:a\ ;\ y) = o:a\ ;\ m:(x + y)$

3. $m:(x + o:(y + z)) = o:(m:(x + y) + m:(x + z))$,

and another one like 3, with the roles of o and m reversed. In 1, m cannot make the choice on the left-hand side and the initiative to do something lies with o. However, m retains initiative as far as the choice is concerned. In 2, o does not choose at all, but simply does a and passes control over the choice to m. It is not obvious why control should be passed to m in 2. Together, 1 and 2 imply that the initiative for a "truly" nondeterministic choice always lies with m. Thus, two forms of nondeterminism are identified: "true" nondeterminism of the $ax + ay$ kind, in which an event leads to an element of a set of possible next states, and lack of control over a choice, as in $\iota_0:(\iota:\alpha_1 + \iota:\alpha_2)$, where ι has no control over the choice. We see nothing wrong in identifying these two forms of nondeterminism, but see no particular reason for making this identification either.

3 says that

Mary chooses between x and giving Otto the choice between y and z

iff

Otto offers Mary the choice between choosing x or y or choosing x or z.

The problem with this is that the order of choices as well as what the actors choose from is different. Hoare [15, pages 107-108] argues for this equation on the grounds that the effect of both sides is the same. We think these conflicting intuitions can be harmonized by simply dropping axiom 3 from the process theory. In our system, we could add

3' $[m:(x + o:(y + z))]\Phi \leftrightarrow [o:(m:(x + y) + m:(x + z))]\Phi$,

as an axiom, expressing Hoare's intuition that the two processes have the same effect, while allowing at the same time for the intuition that the processes themselves are not equal.

5. Conclusion

We gave a semantics and a sound and complete inference system for dynamic logic with equality. To this we added the concept of an actor, which allowed us to express accurately different kinds of nondeterminism, and to distinguish control over and action (including choice) from the visibility of an action. This was compared with several treatments of these concepts in CCS and CSP.

In addition, we included the concept of negated action in dynamic logic, which allowed us to introduce deontic operators in the language. Together with the actor concept, this allows a solution of the long-standing paradox of free choice permission.

The process theory used in this paper is rudimentary and contains no recursive processes or communication. In the future, we wil extend the theory to these processes, and also start work on an operational aspects of the specification language.

6. References

1. al-Hibri, A., *Deontic Logic*, University Press of America (1978).

2. Bergstra, J.A. and Klop, J.W., "Process Algebra for Synchronous Communication," *Information and Control* **60**, pp. 109-137 (1984).

3. Bergstra, J.A. and Klop, J.W., "Algebra of Communicating Processes with Abstraction,"

Theoretical Computer Science **37**, pp. 77-121 (1985).

4. Bergstra, J.A. and Klop, J.W., "Algebra of Communicating Processes," pp. 89-138 in *Mathematics and Computer Science (CWI Monographs 1)*, ed. J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, North-Holland (1986).
5. Castañeda, H.-N., "The Paradoxes of Deontic Logic," in *New Studies in Deontic Logic*, ed. R. Hilpinen, Reidel (1981).
6. Dignum, F.P.M. and Meyer, J.-J.Ch., "Negations of Transactions and Their Use in the Specification of Dynamic and Deontic Integrity Constraints," pp. 61-80 in *Semantics for Concurrency*, ed. M.Z. Kwiatkowska, M.W. Shields, and R.M. Thomas, Springer (1990).
7. Ehrig, H. and Mahr, B., *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics*, Springer (1985). EATCS Monographs on Theoretical Computer Science, Vol. 6.
8. Fiadeiro, J. and Maibaum, T., "Temporal Reasoning over Deontic Specifications," Technical Report, Department of Computing, Imperial College (1989).
9. Føllesdal, D. and Hilpinen, R., "Deontic Logic: An Introduction," pp. 1-35 in *Deontic Logic: Introductory and Systematic Readings*, ed. R. Hilpinen, Reidel (1981).
10. Goguen, J.A., Jouannaud, J.-P., and Meseguer, J., "Operational Semantics for Order-Sorted Algebra," pp. 221-231 in *12th International Coloquium on Automata, Languages and Programming*, ed. W. Brauer, Springer Lecture Notes in Computer Science 194 (1985).
11. Goguen, J.A. and Meseguer, J., "An Order-Sorted Algebra Approach to the Constructors and Selectors Problem," in *Logic in Computer Science* (1987).
12. Goguen, J.A. and Meseguer, J., *Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations*, Programming Research Group, Oxford, and SRI International, Menlo Park (June 19, 1989).
13. Harel, D., "Dynamic Logic," pp. 497-604 in *Handbook of Philosophical Logic II*, ed. D.M. Gabbay and F. Guenthner, Reidel (1984).
14. Hilpinen, R., "Conditionals in Possible Worlds," pp. 299-335 in *Contemporary Philosophy, a New Survey*, ed. G. Fløstad, Reidel.
15. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall (1985).
16. Hoek, W. van der and Meyer, J.J.Ch., "Explicating Some Issues in Implicit Knowledge," Technical Report IR-222, Department of Mathematics and Computer science, Vrije Universiteit, Amsterdam (September 1990).
17. Hughes, G.E. and Cresswell, M.J., *A Companion to Modal Logic*, Methuen (1984).
18. Kalinowski, G., *Einführung in die Normenlogik*, Athenäum Press (1972).
19. Kamp, H., "Free Choice Permission," *Aristotelian Society Proceedings N.S.* **74**, pp. 57-74 (1973-1974).
20. Khosla, S. and Maibaum, T.S.E., "The Prescription and Description of State Based Systems," pp. 243-294 in *Temporal Logic in Specification*, ed. B. Banieqbal, H. Barringer, A. Pnueli, Springer (1987). Lecture Notes in Computer Science 398.
21. Khosla, S., "System Specification: A Deontic Approach," PhD Thesis, Department of

Computing, Imperial College, London (1988).

22. Meyden, R. van der, "The Dynamic Logic of Permission," pp. 72-78 in *Proceedings, 5th IEEE Conference on Logic in Computer Science*, Philadelphia (1990).

23. Meyer, J.-J.Ch., "Free Choice Permissions and Ross's Paradox: Internal vs External Nondeterminism," Report IR-130, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam (august 1987).

24. Meyer, J.-J.Ch., "A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic," *Notre Dame Journal of Formal Logic* **29**, pp. 109-136 (winter 1988).

25. Meyer, J.-J.Ch., "Using Programming Concepts in Deontic Reasoning," pp. 117-145 in *Semantics and Contextual Expression*, ed. R. Bartsch, J.F.A.K. van Benthem, and P. van Emde Boas, FORIS publications, Dordrecht/Riverton (1989).

26. Milner, R., *A Calculus of Communicating Systems*, Springer (1980). Lecture Notes in Computer Science 92.

27. Nicola, R. de and Hennessy, M., "CCS without τ's," pp. 138-152 in *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, ed. H. Ehrig, R. Kowalski, G. Levi and U. Montanari, Springer Lecture Notes in Computer Science 249 (march 1987).

28. Åqvist, L., "Deontic Logic," pp. 605-714 in *Handbook of Philosophical Logic II*, ed. D.M. Gabbay and F. Guenthner, Reidel (1984).

29. Smolka, G., Nutt, W., Goguen, J.A., and Meseguer, J., "Order-Sorted Equational Computation," SEKI Report SR-87-14, Universität Kaiserslautern (December 1987).

30. Wieringa, R.J., Weigand, H., Meyer, J.-J. Ch., and Dignum, F., "The Inheritance of Dynamic and Deontic Integrity Constraints," *Annals of Mathematics and Artificial Intelligence*, To be published.

31. Wieringa, R.J., Meyer, J.-J. Ch., and Weigand, H., "Specifying Dynamic and Deontic Integrity Constraints," *Data and Knowledge Engineering* **4**, pp. 157-189 (1989).

32. Wright, G.H. von, *An Essay in Deontic Logic and the General Theory of Action*, North-Holland (1968).