A note on processes for plan-execution and powerdomains for plan-comparison

Pym, David; Pryor, Louise; Murphy, David

Technical Report No. 719

Department of
Computer Science

A note on processes for plan-execution and powerdomains for plan-comparison

David Pym
Louise Pryor
David Murphy

Queen Mary and Westfield College

1996

A note on processes for plan-execution and powerdomains for plan-comparison

David Pym
Queen Mary & Westfield College
University of London

Louise Pryor
Department of Artificial Intelligence
University of Edinburgh

David Murphy
Securities & Futures Authority
London

Abstract

This paper proposes a representation for plans and actions based on the *algebraic theory of processes*. It is argued that the requirements of plan-execution are better met by representing actions through the *processes by which changes occur* than by the more widely used state-change representation. A simple algebra of plans, based on process-combinators, is described and shown to be adequate for a wide variety of plans. The implications of this type of plan-representation are discussed and its advantages for meta-reasoning (including plan-comparison) outlined. This paper presents the theory of processes from the point of view of planning and describes a novel method of plan-comparison which draws upon ideas in domain theory.

1 Introduction

The execution of plans in unpredictable environments is fraught with difficulty: actions may not have the expected results, the environment may change in unexpected ways, and there may be unforeseen opportunities. Classical theories of planning, in which it is assumed that the world changes only as a result of the planner's deterministic actions, are inadequate in such circumstances. Classical theories of plan-representation are motivated by the requirements of plan construction: how to represent plans in order to facilitate reasoning about them. In this paper, ignoring for the moment the issue of how to construct plans, we concentrate on the requirements placed on the representation of plans by their execution, and on the comparison of plans that have non-deterministic outcomes.

Historically, there have been two main approaches to representing plans: the situation calculus [22] and STRIPS add-and-delete lists [8]. Both these approaches, and others that derive from them, are based on the notion of action as a transition between states [27]. However, instead of asking what actions are required to bring about a given change of state, it is possible to ask what changes have been brought about by a given action. We believe that the latter perspective has many advantages over the former, especially when considering plan-execution, and therefore propose an approach based on the *algebraic theory of processes*, shifting attention from changes of state to the processes by which transitions occur. We contend that a process-based analysis is more powerful and more expressive than those based on state-transitions.

Plan-comparison has traditionally been considered in one of two ways. In classical theories of planning, plans will either succeed or fail, and their outcome is predictable in advance. However, in an unpredictable environment it is impossible to guarantee that a plan will succeed; moreover, there may be gradations of success and failure. These problems have generally been addressed by invoking decision-theoretic notions: a utility value is assigned to each possible outcome of a plan, and the expected utility of the plan then calculated. We propose a generalized notion of plan-comparison that accounts for decision-theoretic results, but is not limited to situations in which utility values can be assigned. Instead of imposing a total order on plan outcomes via their utility values, our analysis, which uses the process-based plan representation we propose, requires only a partial order. Our analysis exploits the powerdomain semantics of process-evolution [15, 29] and Abramsky's ideas about "domain theory in logical form" [2].

In this paper we present the beginnings of a process-theoretic analysis of plan-representation, plan-execution and plan-comparison. Our language of processes will be the algebraic/logical theory due to Milner [25, 15].[1] In § 2, we discuss the advantages that a process-based representation has over a purely state-based representation, based on the relationship between the agent executing a plan and its environment and the requirements that plan-execution places on plan representations. In § 3, we define an algebra of processes, presenting and illustrating a number of combinators that allow us to construct complex plans from simpler ones. In § 4, we describe a method of comparing plans represented in our process algebra.

The use of the process theory to represent plans is not new. It was first suggested by McCarthy and Hayes [22]; more recently, both the Procedural Reasoning System (PRS) of Georgeff and Lansky [11] and Lyons and Hendriks's $\mathcal{RS}$ [20, 21] are based on process theory. Lansky [18] introduces the notion of *action-*

[1] Some of the ideas were discussed in [34]. References [17] and [19] in [34] should be taken to refer to the present paper. Reference [18] in [34] should be taken to refer to [33] in the present paper.

based planning for use in logistical planning domains. Firby's RAPS system [9, 10] and McDermott's RPL [24] are both essentially based on actions. Little attention has, however, been paid to the advantages of using the established results of process theory. For much more discussion see: *J. Logic Computat.* 4(5), 1994.

2 Executing plans

Theories of AI planning have in general concentrated on issues of plan construction and reasoning about plans. They have been able to do ignore issues of plan execution because they assume that the world is *predictable*: that it is possible to foresee accurately and in detail all the circumstances that will arise and the results of all actions. If this is so, plan execution is simple: the specified actions are performed in the correct order; the unexpected cannot occur and nothing can go wrong. Unfortunately, the real world is not like this. Consider, for example, a robot collecting aluminium cans for recycling in an office building. It cannot know in advance where all the cans are: indeed, the people in the office building will put out new ones while the collection is in progress. People may move items of furniture and open and close doors, thus complicating the its navigation task. Moreover, the robot's gripper will not always grasp the cans effectively: sometimes it may knock one over instead of picking it up. Many environments are, like the robot's office bulding, unpredictable. Agents operating in them cannot foresee the results of their actions or what circumstances will obtain in the future. In particular, many environments are *complex* (it is impossible for an agent to maintain or even acquire a faithful representation of the environment), *dynamic* (the environment changes as a result of the actions of other agents and of exogenous influences) and *nondeterministic* (the situation in which an action is performed may not fully determine its effects).

Under these circumstances, it is impractical to expect an agent to use a monolithic, detailed plan that specifies exactly what it should do and that is always guaranteed to work [32]. Instead, we should accept that plans can and do change during execution; that an agent may have different plans for its different tasks, that interact with each other and the environment in unforeseen ways; and that it is important that plans are *operational* inasmuch as they must provide adequate guidance to the agent that is to execute them.

In the remainder of this section we discuss why a process-based representation of plans is effective at handling the interactions between an agent and its environment. The remaining issues pertaining to the representation of plans are addressed in § 3.

2.1 Agents in environments

An important aspect of plan execution is the relationship between an agent executing a plan and the environment in which it operates. The interaction between them is two-way: the agent is affected by the environment through its perceptions and affects the environment through its actions. Moreover, the agent has only limited knowledge of its environment. It can observe and interact with the environment locally but in general it can predict neither the environment's nor its own development. Any theory of planning must therefore recognize the essentially non-privileged status of agents in the world; an agent's own actions should be represented in the same way as external actions.

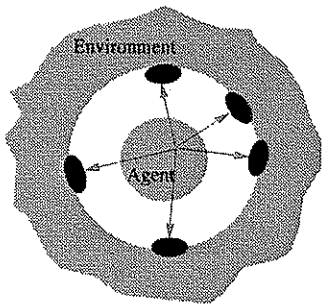

Figure 1: An intensional agent in its environment

We should like our plan-representations to reflect this view of an intensional agent interacting with its environment; in addition, as we noted above, it should account for interactions between plans that are executing concurrently. The *algebraic theory of processes* [15, 25], which deals with the concurrent execution of communicating processes, provides a framework that meets these requirements. In this theory, the possible behaviours of an agent and its environment are specified as *separate* but *communicating* processes. Communication works through the synchronization of pairs of *actions* and *coactions* across processes executing in parallel (see § 3.4). For example, the presence in the environment of a can on a table is recognized by the robot as a coaction $\overline{PickUp}$ which synchronizes with the robot's *PickUp* action, resulting in the transfer of the can from the table to the robot's gripper.

Processes evolve during their execution, the form their evolution takes depending on their interactions with other processes. We can specify the evolutions of processes by giving their operational semantics (see § 3.2). If we represent a plan as a process, its operational semantics thus effectively gives a proof theory of its execution. The joint behaviour of the agent in the environment can then be inferred from those of the individual processes and their interactions. Moreover, the specification of a plan is fully operational: the actions to be performed and the relationships between them, the choices to be made, and how the plan is affected by external events are all explicit in the plan's definition.

We therefore treat agents and environments as two interacting autonomous processes. We visualize

an agent as an explorer, finding out about its environment through the interactions that occur (see Figure 1). The agent can communicate with the environment by perception and action only at restricted locations. All actions other than those performed by the execution agent are represented as environment actions: these may be exogenous events, earthquakes or fire alarms for example, or actions performed by other agents in the world.

In this view, a plan describes a collection of permitted actions performed by the two processes.[2] The agent performs only actions that are permitted by the plan and realizes that the plan is inadequate when the environment does not respond as expected. This means that in a plan we need only specify those actions of the environment that directly affect the current state of the plan; of course, the environment may, and usually will, perform many other actions too, some of which may be relevant later.

The use of process algebras to represent plans has, we believe, several advantages. As well as having a well-developed theory with a rich body of results to draw on, they are more expressive than representations based purely on state-changes, as we discuss in the next section.

2.2 Action representations

Traditionally, plans have been seen as combinations (often, simple sequences) of actions. These actions, which we shall term *AI actions*, are usually considered to be simple, indivisible entities, and are often assumed to be instantaneous. They are generally defined in terms of the state-changes that they produce (see, for example, Pednault's description of the state-transition model of action in [27]). Representing AI actions in this way facilitates reasoning about them when constructing plans. However, it is not optimal when it comes to executing plans. *A priori*, a purely state-based representation is not operational in that it does not specify how the action should be performed. Telling an agent what the result of an action should be is not the same as telling it what the action is. Moreover, when an action is actually performed it may not have the expected results. From the point of view of an agent executing a plan, the results the action was expected to have when the plan was constructed are not important: instead, the important issues are, for example, how the action should be performed, when to stop performing it, and what to do afterwards.

In our system of plan-representation, AI actions are represented as *processes* built out of *basic actions* and process *combinators* (see § 3). Basic actions are the smallest possible building blocks of behaviours, for example the smallest movement that can be made by the actuators of the robot's arm. AI actions may thus have internal structure, need not be instantaneous and are defined in terms of the basic actions required to perform them. The mathematical theory of processes allows us to reason about how they affect the world: we discuss one application in § 4.

Purely state-based representations of AI actions have two major disadvantages: (i) such representations demand that the frame problem be addressed; (ii) they do not handle interruptions to actions adequately. As we shall see, these two issues are linked.

The frame problem is essentially that of determining how an action affects the world. While this is a vital issue when reasoning about actions and plans, it might be thought that it is not important in their execution, thus rendering it unnecessary to represent all possible effects of an action. However, this is not the case. In a complex, dynamic and nondeterministic world it is impossible for plan-execution and plan-construction to be separated entirely. Plan-execution will necessarily involve replanning: *i.e.*, reasoning about what to do next. So even when considering plan plan-execution, plan-construction cannot be ignored.

There are two common approaches to the frame problem in purely state-based approaches: (i) that used in the situation calculus of specifying explicitly (via frame axioms) the propositions that are not affected by the action; (ii) the use of the STRIPS assumption to assume that no effects occur that are not explicitly specified. While we do not claim that the use of a process-based representation will solve the frame problem, we do believe that the issue need not be addressed: it is possible to represent actions and to reason about their effects without being forced to accept the constraints of these two approaches. We substantiate these points in a specific example below.

The second disadvantage of purely state-based action representations is that they treat all actions as being instantaneous. They define actions in terms of a transition between the state in which they are executed (the initial state) and the state resulting from their execution (the final state). Purely state-based representations have nothing to say about intermediate states. Although they are based on the results of actions, they ignore the effects that an action has while it is being executed. This causes problems when an action is interrupted during execution, and replanning has to occur in an unforeseen situation. These disadvantages are also discussed in [6].

Of course, there have been several developments of purely state-based ideas that address these issues. For example, the need for frame axioms can be alleviated by introducing systems of non-monotonic reasoning, such as in [14]; Lin and Shoham [19] have considered concurrent actions in the situation calculus. Alternatively, the event calculus [17] considers the interplay between events (actions) and time. Full discussion of the strengths of these approaches is beyond the scope of this extended abstract, but we contend that they amount to moves towards basing planning on a theory

[2] For simplicity, we shall not differentiate between actions performed by the execution agent and those performed by the environment; it will usually be obvious which we mean.

of processes. Several very useful papers along these lines, too many to discuss in detail here, appeared in: *Journal of Logic and Computation* 4(5), 1994, Special Issue: Actions and Processes. Of particular relevance is the paper by J. van Bentham, J. van Eijck and V. Stebletsova. We propose going all the way to the use of an algebraic theory of processes and exploiting its elegant and substantial theoretical results.

To illustrate the problems of an approach based purely on state-changes, consider our can-collecting robot. Suppose that in addition to collecting aluminium cans, it also fills the office drinking fountain. The drinking fountain consists of a reservoir with a filter, and has a tap at the bottom which enables people to fill their drinking glasses. The robot fills the reservoir from a jug. People may use the drinking fountain while it is performing the action of filling the reservoir. The robot's plan is to keep filling the reservoir until it is full. For simplicity, we assume that the jug contains as much water as necessary.

Note that the level of water in the reservoir varies continuously during the action of pouring from the jug. However, the exact course the level follows depends on the people filling their glasses and is not predictable in advance. If the robot interrupts the pouring action for any reason (a fire alarm, say) the water in the reservoir may be at any level between empty and full. Replanning must then take place: the plan that is chosen will depend on the exact circumstances, but one possibility is to try again to fill the reservoir (possibly it was a false alarm).

It is easy to construct a process-based plan using recursion: "pour water into the reservoir until it is full" (see § 3.3). Constructing a plan is, however, more difficult in a purely state-based representation. It is clearly impossible to define a pouring action in terms of how much water is added to the reservoir: it must be defined in terms of how much water leaves the jug. When replanning, then, the required amount of water must be determined. It is conceivable (though unlikely) that it would be possible to use perception to determine the amount required under the assumption that none is removed during the filling operation. However, this assumption is unreasonable: during the time it takes to fill the reservoir, it is likely that someone will want a drink. In these circumstances, it is impossible to construct a successful plan from pouring actions defined simply in terms of the amount of water to be transferred from the jug. Moreover, a pouring action defined purely in terms of state-changes is not operational: the motor controls of the robot may well be such that it is impossible to specify the exact movements that should be performed in order to result in a given amount of water moving from the jug to the reservoir.

An obvious solution is to construct a plan that involves adding small fixed amounts of water until the reservoir is full, and indeed this would be possible. A plan of this kind amounts, essentially, to an implementation of the process-theoretic solution (see § 3.3), without the benefit of the direct applicability of the *theory* of processes. Roughly speaking, solutions of this kind are produced by developments of purely state-based approaches such as those represented by the event calculus and non-monotonic logics (for many relevant discussions see *J. Logic Computat.* 4(5), 1994).

Finally, note that we are not advocating an approach that does away with reasoning about states altogether: our point is that expressing AI actions in terms of processes is more expressive and enables the use of powerful analytic techniques (see § 4). Moreover, we conjecture that non-monotonic reasoning will play an important rôle in a process-based theory of plan-construction. However, such considerations are beyond the scope of this extended abstract.

3 A process theory of plans

We have seen that the representation of actions as processes has significant advantages over their traditional representation as state-changes; we now consider how basic actions can be combined to form plans. We present a basic syntax of process-combinators that allow us to construct complex plans from actions and simpler plans. We present the operational semantics of the combinators and show, by considering a variety of examples, that our algebra is sufficiently expressive to describe a wide variety of plans.[3]

Plans are represented by processes built up from members of the set of *basic actions*, **Act**, using a set of *process combinators*. The basic machinery we shall use is that of *transition relations*: we write transitions of the form $P \xrightarrow{a} P'$ to indicate that the process P (representing a plan) is capable of performing the action a and, in so doing, becoming the process P'.

3.1 Syntax of processes

We introduce the syntax of processes in four stages, according to the grammar (1) and its progressive extensions (2) – (4) below. We discuss their meaning and use in the sequel. The basic syntax is given in (1). This language consists of the empty plan (returning outcome r), action prefix, external choice and internal choice.

$$P ::= \textsc{Nil}(r) \mid a \cdot P \mid P + P \mid P \oplus P \qquad (1)$$

Here the basic actions a are taken from the set **Act** and outcomes r are taken from a given set **Out**. For now we need make no further assumptions about the set **Out**. Later on, in § 4, we shall require it to carry the structure of an algebraic bounded-complete domain.

[3] Here we shall use the terms "agent" and "plan" interchangeably to denote the behaviour of an agent executing a plan.

Practical applications of this theory require a richer collection of combinators, including recursion,

$$P ::= \mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X})), \tag{2}$$

where $\tilde{E}$ notes a vector of expressions; parallel composition,

$$P ::= P \parallel P \tag{3}$$

and sequential composition,

$$P ::= P\,;\,P\,. \tag{4}$$

This basic syntax of processes can be extended in a number of ways to deal with specific types of situations. For example, information about timing,[4] location, causal relationships between basic actions, probabilities and utilities can be incorporated into the calculus without significantly complicating the metatheory. See, for example, [3, 12].

3.2 Operational semantics

We begin by considering the basic combinators given by equation (1). We give their operational semantics in the usual *natural deduction* style [30, 31]. The meaning of a combinator is determined by the rule of inference that introduces it. All rules are of the form

$$\frac{\textsc{Premiss}_1 \ldots \textsc{Premiss}_m}{\textsc{Conclusion}}\,,$$

with the combinator being defined by the rule occurring only in the conclusion. Axioms are the special case in which the set of premisses is empty.

We discuss the sorts of plans that they can define, which include finite contingent and partially ordered plans. We then consider the extension to the language of plans given in equations (2), (3) and (4), allowing us to express recursive and reactive plans. Here many choices of meaning are possible within the definitional capability of operational semantics.

3.2.1 The empty plan

$\textsc{Nil}(r)$ is the plan that does nothing and returns the outcome r. No transition is possible from the empty plan. Formally, the meaning of this plan is defined by its operational semantics:

$$\textsc{Nil}\quad \frac{r \in \mathbf{Out}}{\textsc{Nil}(r) \not\longrightarrow}$$

Often, r denotes either success ($\top$) or failure ($\bot$); the notion of plan-outcome can, however, be much more general (see § 4).

[4]Note, for example, that our assumption of an interleaving ontology of parallelism yields one notion of simultaneity whereas a non-interleaving assumption would yield another.

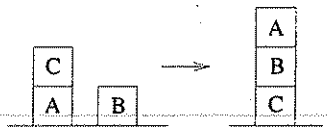

Figure 2: Sussman's anomaly

3.2.2 Action prefix

Our first combinator is the simple *sequence* or *action prefix* combinator. If $a \in \mathbf{Act}$, then $a \cdot P$ is the plan that first performs an action a and then executes the plan P. The meaning of this combinator is given by:

$$\textsc{Pref}\quad \frac{a \in \mathbf{Act}}{a \cdot P \xrightarrow{a} P}$$

With this combinator we can describe all plans that consist of a totally ordered sequence of actions. For example, the well-known plan that solves Sussman's anomaly (shown in Figure 2) is:[5]

$$\mathit{Putdown}(C) \cdot \mathit{Move}(B, C) \cdot \mathit{Move}(A, B) \cdot \textsc{Nil}(\top)$$

Strictly speaking, every plan ends with the empty plan $\textsc{Nil}$ returning a result; we shall usually omit this when the result is $\top$.

So far, we have limited ourselves to plans with no alternative courses of action; they have been *noncontingent* plans. However, plans for use in unpredictable environments often *branch*: there may be several alternative courses of action; the one to be pursued is dictated by the conditions obtaining at the time of execution. For example, a plan to paint a chair the same colour as the table depends on the colour of the table [7]. In order to represent this plan we must be able to represent the environment; we argued above that this should be done by representing it as an agent on the same terms as the planner. In particular, we represent the results of the planner's perceptory actions as actions performed by the environment; the plan to paint the chair is $\mathit{Observe}(\mathit{Table}) \cdot \mathit{Colour}(\mathit{Table}, \mathit{Col}) \cdot \textsc{Paint}(\mathit{Chair}, \mathit{Col})$.

3.2.3 External choice

In the plan to paint the chair we can represent different branches simply through the arguments of the actions. Often, however, different branches of the plan consist of completely different actions. To see this, consider how our can-collecting robot should pick up a can: upright cans should be grasped directly, but a can on its side should be rolled against a fixed object and then grasped. A plan to pick up a can thus depends on the can's orientation, which is decided by the environment. The robot must be prepared for both alternatives. For

[5]Note that our theory permits actions to have arguments

the agent (the robot), this is an *external* choice (not under its control).

This notion of external choice is captured by the combinator $+$. If P and Q are plans, then $P+Q$ is the plan that makes an external choice between plans P and Q. The meaning of $+$ is defined by its operational semantics:

$$\text{CL}\ \frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'} \qquad \text{CR}\ \frac{P \xrightarrow{a} P'}{Q+P \xrightarrow{a} P'}.$$

Using this combinator the robot's plan to pick up a can can be written $\mathit{Look} \cdot (\mathit{Upright} \cdot \textsc{Grasp} + \mathit{OnSide} \cdot \textsc{RollAndGrasp})$.

A more complex plan using this combinator is that produced by CNLP to solve the problem of getting to a ski resort [28]. This is discussed in detail in [33].

3.2.4 Internal choice

The combinator $\oplus$ allows the represention of plans that make *internal* choices without reference to the environment. If P and Q are plans, then $P \oplus Q$ is the plan that makes an internal choice between plans P and Q. As usual, the meaning of this combinator is defined by its operational semantics:

$$\text{NL}\ \frac{}{P \oplus Q \rightarrowtail P} \qquad \text{NR}\ \frac{}{Q \oplus P \rightarrowtail P},$$

where $R \rightarrowtail R'$ denotes that R undergoes an internal transformation into R'. For example, if we were indifferent as to which door our robot uses to enter a room, we could represent its plan as $\textsc{Door1} \oplus \textsc{Door2}$.

The notions of external and internal choice are in a sense duals. When the planner sees an external choice, the environment sees an internal choice and vice versa. For example, suppose the robot tosses a coin to decide which door to use. We represent the robot's plan using the combinator for external choice: $(\mathit{Heads} \cdot \textsc{Door1}) + (\mathit{Tails} \cdot \textsc{Door2})$. A description of the environment, in contrast, models the fact that it is it, rather than the robot, that decides how the coin lands, making the choice without reference to external factors. We therefore represent the process that models the environment using the combinator for internal choice: $(\mathit{Heads} \cdot \textsc{Nil}) \oplus (\mathit{Tails} \cdot \textsc{Nil})$.

These examples show that the evolutions of environments and agents within them cannot be considered independently of one another. We must instead analyse the mutual interactions of processes and environments in order to determine their joint behaviour.

3.3 Recursion

Recursive definitions are an essential aspect of planning. For example, the robot's action of filling the drinking fountain can be described as "continue pouring until the reservoir is full". In process algebra, recursive definitions are achieved via the *fixed point combinator*, $\mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X}))$. This should be read as "the process $\tilde{X}$ such that $\tilde{X} = \tilde{E}(\tilde{X})$". Formally, its meaning is given by its operational semantics:

$$\text{Rec}\ \frac{\tilde{E}[\mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X}))/\tilde{X}] \xrightarrow{a} \tilde{E}'}{\mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X})) \xrightarrow{a} \tilde{E}'}.$$

In the premiss, $\mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X}))$ is substituted for $\tilde{X}$, where $\tilde{X}$ is such that $\tilde{X} = \mu_{\tilde{X}}(\tilde{X} = \tilde{E}(\tilde{X}))$.

Consider, for example, the following recursive process that describes the robot filling the drinking fountain:

$$\textsc{Fill} \ = \ \mu_X(X = ((\mathit{Full} \cdot \textsc{Nil}) + (\mathit{Pour} \cdot X))).$$

If the reservoir is not full, the robot will continue to pour.[6] It is clear that the definition of this plan can easily be modified to take account of more complex circumstances.

Although the recursion combinator $\mu_{\tilde{X}}$ is very convenient, for theoretical purposes there is no need to introduce it at all: a process defined recursively can also be described as a solution to a set of equations not involving the combinator $\mu_{\tilde{X}}$.

3.4 Parallel composition

We consider the combinator for *parallel composition of processes*: $P \parallel Q$ is the process in which the processes P and Q proceed together, either independently or interacting with each other via *action/coaction pairs*.

We begin with an example of independent evolution. In navigating around the office, our can-collecting robot may be simultaneously moving and updating its representation of the office to take account of any changes, such as relocations of furniture, that may have occurred:

$$\textsc{Nav} \ = \ \mathit{Move} \cdot \textsc{Nav} \parallel \textsc{Update}.$$

The plan UPDATE can be a complex process, such as

$$\textsc{Update} \ = \ \mathit{Look} \cdot \mathit{AlterMemory} \cdot \textsc{Nav}.$$

Now consider interacting evolution, via action/coaction pairs. Suppose we include in the definition of navigation a case that describes our robot's act of picking up a can:

$$\textsc{Nav} \ = \ (\mathit{Move} \cdot \textsc{Nav} + \mathit{PickUp} \cdot \textsc{Nav}) \parallel \textsc{Update}.$$

The evolution of the robot, described by its executing NAV, proceeds in parallel with the evolution of the environment, described by a process such as

$$\textsc{Env} \ = \ \ldots + \mathit{MoveDesk} \cdot \textsc{Env} + \overline{\mathit{PickUp}} \cdot \textsc{Env} + \ldots$$

including the coaction $\overline{\mathit{PickUp}}$ that describes the existence of a can which may be collected by our robot.

[6] Often we omit the μ_X, writing, in this example, just $\textsc{Fill} = \mathit{Full} \cdot \textsc{Nil} + \mathit{Pour} \cdot \textsc{Fill}$.

The joint evolution of the environment and the robot can then be described as the evolution of the process ENV ‖ NAV. The operational semantics of ‖ can be given in the usual way. In this case, we can specify the evolution of ENV ‖ NAV in terms of the evolutions of ENV and NAV as follows:[7]

$$\text{Par}\ \frac{\text{ENV} \xrightarrow{\overline{PickUp}} \text{ENV} \quad \text{NAV} \xrightarrow{PickUp} \text{NAV}}{\text{ENV} \parallel \text{NAV} \xrightarrow{\tau} \text{ENV} \parallel \text{NAV}}.$$

Here τ is a *silent* or *perfect* action, so called because it describes the *synchronization* of *PickUp* and $\overline{PickUp}$, entirely *internal* to the process ENV ‖ NAV [15, 25].

Generally, the mathematical semantics of parallel composition is a delicate matter. A number of choices is available even if we assume an interleaving ontology of processes, in which ‖ is definable in terms of +. The key idea is the *expansion theorem* [15, 25]:

Theorem 1 (Milner) *If* $P = \sum_i \alpha_i \cdot P_i$ *and* $Q = \sum_j \beta_j \cdot Q_j$, *then*

$$\begin{aligned} P \parallel Q \quad = \quad & \textstyle\sum_i \alpha_i \cdot (P_i \parallel Q) + \\ & \textstyle\sum_j \beta_j \cdot (Q_j \parallel P) + \\ & \textstyle\sum_{\alpha_i = \overline{\beta_j}} \tau \cdot (P_i \parallel Q_j), \end{aligned}$$

where $\sum_k R_k$ *denotes the finite external sum.* □

It can readily be seen that this theorem gives a reduction of the meaning of parallel composition to that of external choice.[8]

3.5 Relabelling and restriction

Two further combinators are useful in practice: *relabelling* and *restriction*.

Relabelling amounts to the ability to rename actions explicitly (and rename coactions consistently). We write $P[f]$ to denote the process P under relabelling f. For example, if $P = a \cdot Q + \overline{a} \cdot R$ and $f = \{b/a\}$, then $P[f] = b \cdot Q[f] + \overline{b} \cdot R[f]$.

Restriction is more delicate. It provides a way of internalizing specified actions. For example, the agent $(P \parallel Q)\backslash\{a\}$ is the agent $P \parallel Q$ with the action a internalized. This means that the only permitted synchronizations between a and $\overline{a}$ are those that occur between P and Q. In other words, $P \parallel Q$ is prevented from making a- or $\overline{a}$-synchronizations with its environment. Restriction is a conservative extension of the language of basic combinators: $(P \parallel Q)\backslash\{a\}$ can be expressed as a solution to a set of equations not involving restriction.

3.6 Interruptions

During the execution of a plan by an agent in an environment, the environment may perform actions that cause the agent to abort the current plan and execute an entirely different one. For example, if our can-collecting robot is on its rounds in the office when the fire alarm sounds, then it should stop collecting and make its escape.

It is convenient and natural, we judge, to describe this situation by introducing the *interrupt* combinator, written $P \lhd Q$ and pronounced "Q interrupts P". The idea is that the plan $P \lhd Q$ behaves like P until Q does something and then behaves like Q.

We may modify the definition of the plan NAV To describe our robot's behaviour as follows:

$$\begin{aligned} \text{NAV} \quad = \quad & ((Move \cdot \text{NAV} + PickUp \cdot \text{NAV}) \\ & \parallel \text{UPDATE}) \lhd (Alarm \cdot \text{EMERGENCYEXIT}). \end{aligned}$$

The operational semantics of $\lhd$ can be described in the usual way:

$$\lhd \text{L}\ \frac{P \xrightarrow{a} P'}{P \lhd Q \xrightarrow{a} P' \lhd Q} \qquad \lhd \text{R}\ \frac{Q \xrightarrow{b} Q'}{P \lhd Q \xrightarrow{b} P \lhd Q'}.$$

The theory of interrupt combinators can be found in [25]. Note that interrupt cannot be defined by equations in terms of the basic combinators.

3.7 Sequential composition

The execution monitoring we saw in the last section has much in common with the ideas of so-called "reactive planning". We have shown in [33] that our algebra of plans is adequate to describe the traditional plans produced by classical planning systems, including partially ordered plans; it can also be used to describe reactive plans.

Consider, for example, Agre and Chapman's Pengi [4], which has many different *reaction rules*, each of which is appropriate under different circumstances. There is also an arbitration mechanism that decides between rules if there are several that are applicable. The process that describes Pengi thus takes the form

$$\begin{aligned} \text{PENGI} \quad = \quad & \mu X(((Cond_1 \cdot Action_1 + \\ & \ldots + Cond_n \cdot Action_n) \\ & \parallel \text{ARBITRATION}) \,;\, X), \end{aligned}$$

where the ; combinator denotes the *sequential composition* of processes.[9] Again, we have a recursive definition to represent the execution cycle of repeated rule application.

The operational semantics of this notion of sequential composition can be given simply, although at some length, in the usual natural deduction style illustrated above [25]. Informally, the Q in $P \,;\, Q$ can proceed as soon as P has terminated. Sequential composition can be defined in terms of ‖ [25].

[7] The general form of the rule follows a similar pattern.

[8] This relies upon the representability of finite processes in a *standard form* [15, 25].

[9] Hitherto, we have considered only the case of a process following a basic action; here, we consider the case of a process following another process.

4 Reasoning about plans

We have not yet considered how to reason about plans. There are many issues that arise in this context that are absent from plan-execution: in particular, the possible effects of actions become important.

As we discussed in § 2, one of the advantages of our representation is that the use of processes allows us to ignore the potential effects of actions, which are not required for plan-execution. Clearly, a complete theory of planning cannot ignore plan-construction and modification: they are vital capabilities for any agent operating in an essentially unpredictable world [32]. We believe that the large body of theoretical results provided by the use of process theory gives a sound basis for the necessary analysis.

There are two types of reasoning about actions and plans that must be considered: (i) as designers of agents, we must be able to reason about the agents we design and their interactions with their environments; (ii) an agent must itself be able to reason about its behaviour and that of its environment if it is to adapt effectively to the circumstances in which it finds itself. We restrict our attention to the former.

4.1 Executing and comparing plans

We have discussed how it is that when an agent executes a plan it evolves jointly with its environment. We can describe such a joint evolution as a sequence of the form

$$(E_0, P_0) \stackrel{a_0}{\Longrightarrow} (E_1, P_1) \stackrel{a_1}{\Longrightarrow} \cdots \stackrel{a_{n-1}}{\Longrightarrow} (E_n, P_n),$$

where at each step an action a_i — performed either by the agent (plan) or the environment — occurs.

Note that we have restricted our attention here to the finite evolutions of terminating plans. Generalization to the infinitary evolutions of non-terminating of plans is possible, but would serve here only to complicate our exposition. Note also that we restrict our attention to the basic syntax of plans (1). It should be clear from the remarks in § 3 that this is sufficient for the theoretical purposes of this section: with the exception of interrupts, the other combinators can be expressed in terms of the basic ones.[10]

The interaction of plans and environments, in the notation of the last section, is given in Figure 3. If a silent move is possible by either party then it is possible by the composite (rules IP and IE) since internal moves are private. Synchronizations must be agreed upon, so as to form an action/coaction pair, by the plan and the environment (rule Sync), reflecting the fact that the planner can only do what the environment allows. (A form of synchronization using Hoare's conjunction combinator [25] is also possible.)

[10]The absence of interrupts is a rather minor deficiency. Synchronizations across $\|$ will usually suffice, given some coding effort.

$$\text{IP}\ \frac{P \rightarrowtail P'}{(E,P) \Longrightarrow (E,P')} \qquad \text{IE}\ \frac{E \rightarrowtail E'}{(E,P) \Longrightarrow (E',P)}$$

$$\text{Sync}\ \frac{P \stackrel{a}{\longrightarrow} P' \quad E \stackrel{\overline{a}}{\longrightarrow} E'}{(E,P) \Longrightarrow (E',P')} \qquad \frac{P \stackrel{\overline{a}}{\longrightarrow} P' \quad E \stackrel{a}{\longrightarrow} E'}{(E,P) \Longrightarrow (E',P')}$$

Figure 3: The operational semantics of evolution

Since our syntax of plans includes combinators for non-deterministic choice, a model of all possible executions of a given plan in a given environment carries the structure of a tree, with each possible choice in an execution being interpreted by a branch of the tree.

Definition 2 *A* domain of outcomes, **Out**, *consists in a set O that carries an algebraic bounded-complete partial order, $\sqsubseteq$, including least and greatest elements, $\bot$ (complete failure) and $\top$ (complete success), with respect to $\sqsubseteq$.* □

For technical reasons, we must suppose that we can identify a set $\mathcal{E}_T \times \mathcal{P}_T$ of environment/process pairs that is terminal. A joint evolution of a given plan in a given environment will terminate as an element (E, P) of this set. The outcome of this evolution is then taken to be an assignment of an element of **Out** to (E, P).

Definition 3 *Let $\mathcal{E} \times \mathcal{P}$ be the set of all environment/process pairs, let $\mathcal{E}_T \times \mathcal{P}_T \subseteq \mathcal{E} \times \mathcal{P}$ be a set of terminal environment/process pairs and let* **Out** *be a domain of outcomes. An* **Out***-valued outcome predicate is a partial function $\mathcal{O} : \mathcal{E} \times \mathcal{P} \to$* **Out** *such that $\mathcal{O}(E,P)\downarrow$ (i.e., $\mathcal{O}(E,P)$ is defined) just in case $(E,P) \in \mathcal{E}_T \times \mathcal{P}_T$. An execution $(E_0,P_0) \stackrel{=}{\Longrightarrow} \ldots \stackrel{=}{\Longrightarrow} (E_n,P_n)$ is said to have outcome r if $\mathcal{O}(E_n,P_n)\downarrow = r$. We write $\mathcal{R}(E_0,P_0)$ for the set of all possible outcomes of the plan P_0 in the environment E_0:*

$$\begin{aligned}\mathcal{R}(E_0,P_0) \quad := \quad \{ r \mid \exists (E_n,P_n) \,.\, (E_0,P_0) \stackrel{=}{\Longrightarrow} \\ \ldots \stackrel{=}{\Longrightarrow} (E_n,P_n) \text{ and} \\ \mathcal{O}(E_n,P_n)\downarrow = r \}\end{aligned}$$

□

Note that in decision-theoretic planning **Out** is the set of possible utility values, with the outcome r of executing a plan P from an initial state E_0 being the utility $\mathtt{U}(E_n,P_n)$ associated with the world state (E_n,P_n) resulting from that execution [13]. Plans are then compared on the basis of their possible outcomes by weighting the utilities with the probability that executing the plan will result in the associated state, giving the expected utility of plan P_0 with respect to environment E_0:

$$\mathtt{EU}(E_0,P_0) = \sum_{(E_n,P_n)\in\mathcal{E}_T\times\mathcal{P}_T} \mathtt{P}((E_n,P_n)|(E_0,P_0))\mathtt{U}(E_n,P_n).$$

Note that this gives a totally ordered $\mathcal{R}(E_0,P_0)$, namely a subset of the reals. With our more general notion of outcome, we cannot use probabilities to combine outcomes, and moreover have no total order on

Out. (We remark that probabilistic process calculi are available, although they are beyond the scope of this extended abstract.) Instead, we must extend our notion of ordering to the *sets* of outcomes $\mathcal{R}(E_0, P_0)$. This requires a *powerdomain* [15, 29] on **Out**.

Definition 4 *Let* $\mathbf{Out} = (O, \sqsubseteq)$ *be a domain of outcomes. Three powerdomains, each with underlying set the finite nonempty subsets of* O*, are defined by taking the following orderings:*

1. *The upper powerdomain,* $\wp_u(\mathbf{Out})$*:* $R \sqsubseteq_u S$*, for each* $R, S \in \wp(O)$*, iff, for all* $s \in S$*, there exists an* $r \in R$ *such that* $r \sqsubseteq s$ *(i.e., for everything in* S *there is an element smaller than it in* R*);*
2. *The lower powerdomain,* $\wp_l(\mathbf{Out})$*:* $R \sqsubseteq_l S$ *iff, for all* $r \in R$*, there exists an* $s \in S$ *such that* $r \sqsubseteq s$ *(i.e., for everything in* R *there is an element bigger than it in* S*);*
3. *The convex powerdomain,* $\wp_c(\mathbf{Out})$*:* $R \sqsubseteq_c S$ *iff both conditions (1) and (2) hold.* □

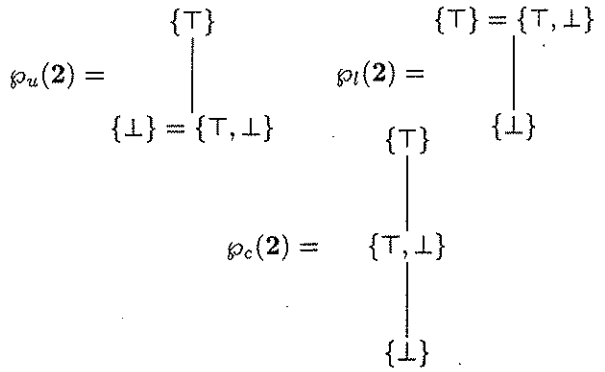

Figure 4: Some powerdomains of **2**

The powerdomain constructions for the two point domain **2** are illustrated in Figure 4: it can be seen that the upper powerdomain corresponds to a pessimistic view of the world (identifying the possibility of failure $\{\top, \bot\}$ with the certainty of it $\{\bot\}$), the lower to an optimistic view (certain success is identified with possible success) whereas the convex powerdomain takes a more balanced view. These intuitions broadly carry over to more complicated domains **Out**. The interested reader is referred to [29, 15, 35, 36], where details supporting the claim that these constructions are "natural" are also given. We are now in a position to compare plans.

Definition 5 *Two plans,* P *and* Q *are said to be i-ordered, written* $P \leq_i Q$*, with respect to a class of environments* **Env** *if and only if, for all* $E \in \mathbf{Env}$*,*

$$\mathcal{R}(E, P) \sqsubseteq_i \mathcal{R}(E, Q).$$

The associated notions of equivalence will be written $\approx_i$ *(i.e.,* $P \approx_i Q$ *if and only if* $P \leq_i Q$ *and* $Q \leq_i P$*). The equivalence* $\approx_c$*, induced by the convex powerdomain* $\wp_c(\mathbf{Out})$*, will play a central rôle in the sequel; it will be referred to as* outcome-equivalence. □

Using these definitions and the operational semantics given in Figure 3, we can deduce $\mathcal{R}(E, P)$ directly from the syntax of E and P and the outcome-predicate. Thus we can determine whether $P \leq_i Q$. Finally, note that the combinators are well-behaved with respect to plan-comparison [15].

Proposition 6 *The basic plan-combinators are all compositional with respect to the testing preorders* $\leq_i$*.* □

4.2 A logic of plan-equivalence

It is useful for us, as designers of agents and, sometimes, environments, to be able to specify features of the behaviour of plans.

Finding the right logic with which to specify and reason about plans is problematic; indeed, the literature is considerable, *e.g.*, [5, 22, 23], but we shall be concerned (i) that the logic be powerful enough to specify the behaviour of plans in the presence of internal and external choice, (ii) that logical equivalence (which identifies plans that model the same formulae) coincide with outcome-equivalence, and (iii) that this coincidence should not be accidental: it should reflect the structure of formulae and of process terms.

One logic satisfying these requirements arises from a consideration of the powerdomain $\wp_c(\mathbf{Out})$.[11]

Consider the following BNF for a (constructive) modal **Out**-valued logic:

$$\phi \quad ::= \quad r \mid \phi \wedge \psi \mid \bigvee_j \phi_j \mid \langle a \rangle \phi \mid [a]\phi$$

and let Φ denote the set of well-formed formulae determined by this grammar. The idea is that r is a basic truth value (so if we were working with **2**, we would just have truth and falsity) and the two logical connectives $\wedge$ (conjunction) and $\bigvee_j$ (j-indexed disjunction) are as usual. The interesting formulae, then, are the modal ones of the form $\langle a \rangle \phi$ or $[a]\phi$. The formula $\langle a \rangle \phi$ tests whether an a change is possible next and ϕ then holds, so the result of $P \models \langle a \rangle \phi$ is certain failure, $\{\bot\}$, if P cannot do an a, and the union of the results of $P' \models \phi$ for all a-derivatives P' of P otherwise. $[a]\phi$ is similar, except that we succeed $\{\top\}$ rather than fail if an a is not possible. It should clear that this logic is a relative of dynamic logic.

We can recast the interaction of the process and the environment by replacing the environment by a formula that describes it. The extent to which a process models such a formula is then a measure of the

[11] Here we are really doing "domain theory in logical form" [2, 36]. A general account would involve defining a domain equation underlying this situation and showing how the Hennessy-Milner-like [16] result we obtain (Proposition 8) arises from solutions to this this equation.

$$
\begin{aligned}
\mathcal{L}(r,P) &= \{r\}- \\
\mathcal{L}(\phi\wedge\psi,P) &= \mathcal{L}(\phi,P)\cap\mathcal{L}(\psi,P) \\
\mathcal{L}\left(\bigvee_j \phi_j, P\right) &= \bigcup_j \mathcal{L}(\phi_j,P) \\
\mathcal{L}(\langle a\rangle\phi,P) &= \bigcup_{P',P\xrightarrow{a}P'} \mathcal{L}(\phi,P')\cup\{\bot \mid P\rightarrowtail\!\!\!\!\!\rightarrow^{*} P'\not\xrightarrow{a}\} \\
\mathcal{L}([a]\phi,P) &= \bigcup_{P',P\xrightarrow{a}P'} \mathcal{L}(\phi,P')\cup\{\top \mid P\rightarrowtail\!\!\!\!\!\rightarrow^{*} P'\not\xrightarrow{a}\}
\end{aligned}
$$

Figure 5: The relationship between the logic and outcome powerdomains

outcome of executing P in the environment modelled by the formula. This idea suggests defining a function

$$\mathcal{L} : \Phi \times \mathcal{P} \to \wp_c(\mathbf{Out})$$

giving the extent to which P is a model for ϕ (Figure 5).

Consider this function in more detail: if we are just working with $\mathbf{Out} = \mathbf{2}$, then $\mathcal{L}(\phi, P)$ can be either $\{\top\}$ (P always guarantees ϕ), $\{\bot\}$ (P never guarantees ϕ) or $\{\top, \bot\}$ (P sometimes but not always guarantees ϕ). This is the logical counterpart of $\mathcal{R}$: instead of asking what outcomes are possible if we run P in E, we ask what outcomes are possible of requiring that P behave like ϕ [26].

The definition of $\mathcal{L}$ given in Figure 5 matches the given intuition for the logic above: the outcome of asking that P model r is just the (singleton) set of outcomes $\{r\}$; that of asking that P model a conjunction or disjunction is the corresponding conjunction or disjunction of outcomes (as an operation on $\wp_c(\mathbf{Out})$). Of more interest are $\mathcal{L}(\langle a\rangle\phi, P)$ and $\mathcal{L}([a]\phi, P)$: the outcomes of asking that P do an a and then behave like ϕ are the set of outcomes of asking that P' can behave like ϕ for all a-derivatives P' of P, together with total failure ($\bot$) if P cannot do an a; the clause for $[a]\phi$ is similar, except that we succeed rather than fail if P cannot do an a. This logic can be used to specify the properties of the plans in [33].

With $\mathcal{L}$, then, we have defined a notion of "models" valued on $\wp_c(\mathbf{Out})$, rather than just $\mathbf{2}$:

$$[\![P \models \phi]\!]^{\mathcal{L}} = \mathcal{L}(\phi, P).$$

This notion naturally suggests a notion of process-equivalence: two processes should be logically equivalent if they have the same behaviour as models.

Definition 7 *Processes P and Q are said to be logically equivalent wrt a class L of formulae L iff, for all formulae $\phi \in L$, $[\![P \models \phi]\!]^{\mathcal{L}} = [\![Q \models \phi]\!]^{\mathcal{L}}$.* □

It is possible to show, by induction on the structure of processes and propositions, that $\mathcal{R}(E, P)$ coincides exactly with $\mathcal{L}(E, P)$. The proof (*cf.* [1]) is quite long and delicate and is omitted from this extended abstract.

Proposition 8 *Two processes are logically equivalent if and only if they are outcome-equivalent.* □

References

[1] S. Abramsky, Observation equivalence as a testing equivalence, Theoretical Computer Science 53, 225-241, 1987.

[2] S. Abramsky, *Domain theory in logical form*, Annals of Pure and Applied Logic 51, 1–77, 1989.

[3] L. Aceto and D. Murphy, *Timing and Causality in Process Algebra. Acta Informatica* (to appear).

[4] P. Agre and D. Chapman, *An implementation of a theory of activity*, Proc. AAAI-87, 1987.

[5] J. Allen, *Towards a general theory of action and time*, Artificial Intelligence 23, 123–154, 1984.

[6] J. Allen and G. Ferguson. *Actions and Events in Interval Temporal Logic*. J. Logic Computat 4(5), 531-579, 1994.

[7] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson, *An approach to planning with incomplete information*, Proc. 3rd Int. Conf. Knowledge Representation and Reasoning, Boston, MA, Morgan Kaufmann, 115–125, 1992.

[8] R. Fikes and N. Nilsson, *STRIPS: a new approach to the application of theorem proving to problem solving*, Artificial Intelligence 2, 189–208, 1971.

[9] R. J. Firby, *Adaptive execution in complex dynamic worlds*, Technical Report 672, Computer Science Department, Yale University, 1989.

[10] R. J. Firby, *Task networks for controlling continuous processes*, Proc. 2nd AIPS, AAAI Press, 49–54, 1994.

[11] M. Georgeff and A. Lansky, *Procedural knowledge*, Proc. IEEE 74(10), 1383–1398, 1986.

[12] R. van Glabbeek, B. Steffen and C. Tofts. *Reactive, generative and stratified models of probabilistic processes*, Proc. 5th IEEE LICS Symposium, Philadelphia, USA, IEEE Computer Society Press, 130–141, 1990.

[13] P. Haddawy and S. Hanks, *Issues in Decision-Theoretic Planning: Symbolic Goals and Numeric Utilities*, Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA, DARPA, 48–58, 1990.

[14] S. Hanks and D. McDermott. *Nonmonotonic logic and temporal projection*. Artificial Intelligence 33(3), 379–412, 1987.

[15] M. Hennessy, *An algebraic theory of processes*, MIT Press, 1988.

[16] M. Hennessy and R. Milner, *Algebraic laws for nondeterminism and concurrency*, J. ACM 32(1), 137–161, 1985.

[17] R. Kowalski and M. Sergot, *A logic-based calculus of events*, New Generation Computing 4(1), 67–95, 1986.

[18] A. Lansky, *Action-based planning*, Proc. 2nd AIPS, Chicago, IL, AAAI Press, 110–115, 1994.

[19] F. Lin and Y. Shoham. *Concurrent actions in the situation calculus*. Proc. AAAI-92, 590–695, 1992.

[20] D. Lyons, *Representing and analysing action plans as networks of concurrent processes*, IEEE Trans. Rob. & Aut. 9(3), 241–256, 1993.

[21] D. Lyons and A. Hendriks, *Exploiting patterns of interaction to achieve reactive behavior*, Artificial Intelligence 73, 117–148, 1995.

[22] J. McCarthy and P. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*, Machine Intelligence 4 (B. Meltzer and D. Michie, Eds.). Edinburgh University Press, 463–502, 1969.

[23] D. McDermott, *A temporal logic for reasoning about processes and plans*, Cognitive Science 6(2), 101–155, 1982.

[24] D. McDermott, *A reactive plan language*, Technical Report 864, Computer Science Dept., Yale University, 1991.

[25] R. Milner, *Communication and concurrency*, Prentice Hall International, 1989.

[26] D. Murphy, *Testing, betting and timed true concurrency*, Proceedings of Concur, 527, Springer-Verlag LNCS, 1991.

[27] E. P. D. Pednault, *ADL and the state-transition model of action*, J. Logic Computat. 4, 467–512, 1994.

[28] M. Peot and D. Smith, *Conditional Nonlinear Planning*, Proc. 1st AIPS, Morgan Kaufmann, 189–197, 1992.

[29] G. D. Plotkin, *A powerdomain construction*. SIAM J. Computing 5, 452–487, 1976.

[30] G. D. Plotkin, *A structural approach to operational semantics*, Technical Report DAIMI-FN-19, Computer Science Department, Århus University, 1981.

[31] D. Prawitz, *Natural Deduction: A Proof-Theoretical Study*, Almqvist & Wiksell, Stockholm, 1965.

[32] L. Pryor, *Opportunities and planning in an unpredictable world*, Technical Report 53, Institute for the Learning Sciences, Northwestern University, 1994.

[33] L. Pryor, D. Pym and D. Murphy, *Processes for plan-execution*, Working Notes of the 14th Workshop of the UK Planning and Scheduling Special Interest Group (S. Steel, editor), University of Essex, 1995. Also available as Technical Report No. 718, Department of Computer Science, Queen Mary and Westfield College, University of London.

[34] D. Pym, L. Pryor and D. Murphy, *Actions as processes: a position on planning*, Proceedings of the AAAI Spring Symposium on Extending Theories of Action: Formal Theory and Practical Applications, Stanford University, 1995. Also available as Technical Report No. 696, Department of Computer Science, Queen Mary and Westfield College, University of London.

[35] S. Vickers, *Topology via logic*, Tracts in Theoretical Computer Science 5, CUP, 1989.

[36] G. Winskel, *Powerdomains and modality*, Theoretical Computer Science 34, 127–137, 1985.