Processes for plan-execution

Pryor, Louise; Pym, David; Murphy, David

Technical Report No. 718

Department of Computer Science

Processes for plan-execution

Louise Pryor
David Pym
David Murphy

Queen Mary and Westfield College

1996

In: Proc. Working Notes of the 14th Workshop of the UK Planning and Scheduling Special Interest Group (S. Steel, ed.), University of Essex, 1995.

Processes for plan-execution

Louise Pryor
Dept. of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
louisep@aisb.ed.ac.uk

David Pym
Queen Mary & Westfield College
University of London
Mile End Road
London E1 4NS
pym@dcs.qmw.ac.uk

David Murphy
Risk Assessment Group
Securities & Futures Authority
Cottons Centre
Cottons Lane
London SE1 2QB

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 the plans found in the literature. The implications of this type of plan-representation are discussed and its advantages for meta-reasoning (including plan-comparison and plan-construction) outlined.

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 questions of how to construct and reason about plans, we concentrate on the requirements placed on the representation of plans by their execution.

Historically, there have been two main approaches to representing plans: the situation calculus [21] and STRIPS add-and-delete lists [7]. Both these approaches, and others that derive from them, are based on the notion of action as a transition between states [25]. 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.

In this paper, we present a system of plan-representation based on the algebraic theory of processes.[1] In § 2, we discuss the advantages that a process-based representation has over a state-based representation, based on the relationship between the agent executing a plan and its environment and the requirements that plan-execution places on

[1] Some of the ideas were briefly discussed in [32]. Reference [18] in [32] should be taken to refer to the present paper. References [17] and [19] in [32] should be taken to refer to [33] in the present paper. The main text of the present paper appeared under the same title in the Working Notes of the 14th Workshop of the UK Planning and Scheduling Special Interest Group (S. Steel, editor), University of Essex, 1995.

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. Finally, we briefly consider some of the implications of this method of plan representation.

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 have the following characteristics:

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;

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 [29]. 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 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.

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* [14, 24], which deals with the concurrent execution of communicating processes, provides a framework that meets these re-

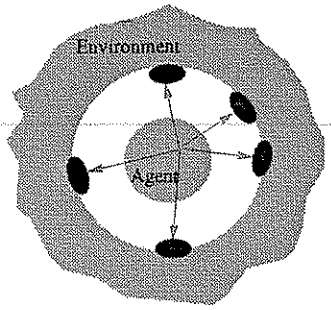

Figure 1: An intensional agent in its environment

quirements. 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 [25]). 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 state-based representation is not operational in that it does not specify how the

[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.

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 this briefly 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, plan modification and reasoning about what to do next. Even when considering plan representations from the point of view of plan-execution, then, issues of plan-construction cannot be ignored.

There are two common approaches to the frame problem in 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 it means 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 philosophically poorly motivated or computationally undesirable constraints of these two approaches.[3] We substantiate these points in a specific example below.

The second disadvantage of 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). 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.

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 [13]; Lin and Shoham [17] have considered concurrent actions in the situation calculus. Alternatively, the event calculus [15] 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 con-

[3]The frame axioms of the situation calculus are philosophically poorly motivated and computationally undesirable. The STRIPS assumption is also philosophically poorly motivated. It is computationally desirable at the cost of expressivity.

tend that they amount to moves towards basing planning on a theory 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, and without loss of generality, 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: it is therefore philosophically unacceptable to use axioms to describe this behaviour. Moreover, even if such descriptions were available, in general their complexity would necessarily make them computationally undesirable. 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 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).

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).

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.[4]

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.

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

Here the basic actions a are taken from the set **Act** and outcomes r are taken from a given set **Out**.

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

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

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

$$P ::= P \parallel P \quad (3)$$

and sequential composition,

$$P ::= P ; P . \quad (4)$$

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,[5] location, causal relationships between basic actions, probabilities and utilities can be incorporated into the calculus without significantly complicating the metatheory. See, for example, [1, 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 [27, 28]. 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.

[4] For now, we shall use the terms "agent" and "plan" interchangeably to denote the behaviour of an agent executing a plan.

[5] 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.

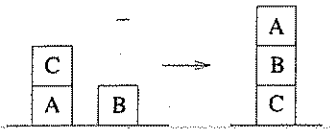

Figure 2: Sussman's anomaly

3.2.1 The empty plan

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:

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

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

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:

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

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:[6]

$$\mathit{Putdown}(C) \cdot \mathit{Move}(B, C) \cdot \mathit{Move}(A, B) \cdot \text{NIL}(\top)$$

Strictly speaking, every plan ends with the empty plan 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. Recent systems within the classical paradigm that construct such plans include Cassandra [31, 30], CNLP [26], SENSp [6] and C-BURIDAN [5]. For example, a plan to paint a chair the same colour as the table depends on the colour of the table [6]. 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 $\mathit{Observe}(\mathit{Table}) \cdot \mathit{Colour}(\mathit{Table}, \mathit{Col}) \cdot \text{PAINT}(\mathit{Chair}, \mathit{Col})$.

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 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 $\mathit{Look} \cdot (\mathit{Upright} \cdot \text{GRASP} + \mathit{OnSide} \cdot \text{ROLLANDGRASP})$. This

[6] Note that our theory permits actions to have arguments.

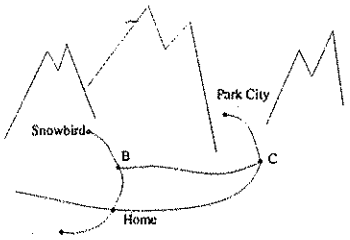

Figure 3: Getting to a ski resort

plan is typical of those constructed by the contingent planners mentioned above.

A more complex plan using this combinator is that produced by CNLP to solve the problem of getting to a ski resort. There are assumed to be two: Snowbird and Park City. The roads are as shown in Figure 3 [26]. A road cannot be driven along if it is covered with snow: the only roads that might be blocked are those leading into the two resorts. The plan that CNLP produces is to drive to B and observe the road leading up to Snowbird; if it is blocked, drive to C and observe the road leading up to Park City, otherwise drive to Snowbird; if the second road is also blocked, the plan fails. In our representation, the plan is:

$$\begin{array}{l} \mathit{Goto}(B) \cdot \mathit{Observe}(B,S) \cdot \\ \quad (\mathit{Clear}(B,S) \cdot \mathit{Goto}(S) \cdot \textsc{Nil}(\top) \\ \qquad + \ \mathit{Blocked}(B,S) \cdot \mathit{Goto}(C) \cdot \mathit{Observe}(C,P) \cdot \\ \qquad\quad (\mathit{Clear}(C,P) \cdot \mathit{Goto}(P) \cdot \textsc{Nil}(\top) \\ \qquad\qquad + \ \mathit{Blocked}(C,P) \cdot \textsc{Nil}(\bot))) \end{array}$$

This plan differs from those we have seen previously in that it is not guaranteed to achieve the goal of getting to a ski resort. If both roads are blocked, there is no route by which a ski resort can be reached, so the goal cannot be achieved. In this case, the plan ends with $\textsc{Nil}(\bot)$, signifying failure. Our representation thus provides a natural way to indicate plan termination, and moreover provides a facility for indicating the expected result of plan-execution.

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 $\textsc{Door1} \oplus \textsc{Door2}$.

This situation is not uncommon: there may be several potential courses of action, all of which are possible. In the plans produced by Cassandra, for example, as long as the conditions that ensure the success of a branch are met, it may be pursued [31]. Consider the plan constructed by Cassandra for driving to Evanston. There are two possible routes, via Ashland or via Western. If the traffic is bad, the route via Western should not be attempted; the route via Ashland is always possible, but it is slower if the traffic is bad. The full plan is

$$\begin{array}{l} \mathit{CheckTraffic} \cdot (\mathit{Jam} \cdot \textsc{Ashland} + \\ \quad \mathit{Clear} \cdot (\textsc{Ashland} \oplus \textsc{Western})) \end{array}$$

This plan always achieves the goal of reaching Evanston. If the traffic is clear, the choice between Western and Ashland is left to the execution agent, which may decide on the slower but more interesting route via Ashland.

The notion of external choice is in a sense the dual of internal choice. 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. It cannot influence whether the coin comes down heads or tails. From the robot's point of view, the choice

is made by the environment; once the environment has acted, by making the coin come down heads or tails, its course of action has been decided. 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. The environment makes this 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.

3.2.5 Derived combinators

Before proceeding with our systematic account of the combinators, we give an example of the flexibility of our analysis. In processes described without the choice combinators $+$ and $\oplus$, the sequence of actions performed is totally ordered by $\cdot$, the sequence combinator. However, many "classical" planners, such as TWEAK [3] and SNLP [20], produce plans having a weaker order on their actions. We now show how weaker orderings can be recovered via the choice combinators.

Consider, for example, Moore's bomb-in-the-toilet problem, described by McDermott [23]. There are two identical packages, one of which contains a bomb. Bombs can be defused by dunking them in water. There is no way of telling which package contains the bomb, so to guarantee safety both packages must be dunked. However, the order in which they are dunked does not matter; moreover, it does not matter whether both packages are carried to the toilet before either is dunked, or whether the first is dunked before the second is carried. The only ordering that is imposed is that each package must be carried before it is dunked. With an internal choice between all possible orderings, the plan can therefore be written[7]

$$
\begin{array}{l}
(\mathit{Carry}(A) \cdot \textsc{Dunk}(A) \cdot \mathit{Carry}(B) \cdot \textsc{Dunk}(B)) \oplus \\
(\mathit{Carry}(B) \cdot \textsc{Dunk}(B) \cdot \mathit{Carry}(A) \cdot \textsc{Dunk}(A)) \oplus \\
(\mathit{Carry}(A) \cdot \mathit{Carry}(B) \cdot \textsc{Dunk}(A) \cdot \textsc{Dunk}(B)) \oplus \\
(\mathit{Carry}(B) \cdot \mathit{Carry}(A) \cdot \textsc{Dunk}(A) \cdot \textsc{Dunk}(B)) \oplus \\
(\mathit{Carry}(A) \cdot \mathit{Carry}(B) \cdot \textsc{Dunk}(B) \cdot \textsc{Dunk}(A)) \oplus \\
(\mathit{Carry}(B) \cdot \mathit{Carry}(A) \cdot \textsc{Dunk}(B) \cdot \textsc{Dunk}(A)) .
\end{array}
$$

It should be clear that we can represent all "classical" plans using this technique, albeit somewhat clumsily. We could therefore, for example, introduce a *derived combinator*, $\bowtie$, as syntactic sugar for this construction. Using this combinator, the plan would be written

$$(\mathit{Carry}(A) \cdot \textsc{Dunk}(A)) \bowtie (\mathit{Carry}(B) \cdot \textsc{Dunk}(B)) .$$

It should be clear that this use of internal choice enables us to eliminate any ordering constraints that we wish from our plans. For example, whenever we want to allow either action a to precede action b or *vice versa* prior to the execution of P, we simply include $(a \cdot b \cdot P) \oplus (b \cdot a \cdot P)$ in the definition of the plan. In the example above, each $\oplus$-component represents a weakening of the ordering constraint on the sequence of actions. Note that one could organize the notation so that all orderings were possible unless specifically prohibited, with many choices in between.

We have now discussed the basic combinators and shown how through their use it is possible to represent all totally-ordered, partially-ordered and contingency plans. We now move on to more advanced combinators that allow us to represent recursion (and hence loops), parallel composition, execution monitoring and reactive plans.

[7] The original point of this problem was to demonstrate that classical planners could not find the correct plan; more recently, planners have been developed that can handle this situation [4].

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:

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

Note that 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.[8] It is clear that the definition of this plan can easily be modified to take account of more complex circumstances.

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.

The joint evolution of the environment and the robot can then be described as the evolution of the process

$$\textsc{Env} \parallel \textsc{Nav}.$$

It remains to give the operational semantics of $\parallel$. In this case, we can specify the evolution of Env $\parallel$ Nav in terms of the evolutions of Env and Nav as follows:[9]

$$\text{Par} \quad \frac{\textsc{Env} \xrightarrow{\overline{\mathit{PickUp}}} \textsc{Env} \quad \textsc{Nav} \xrightarrow{\mathit{PickUp}} \textsc{Nav}}{\textsc{Env} \parallel \textsc{Nav} \xrightarrow{\tau} \textsc{Env} \parallel \textsc{Nav}}.$$

Here τ is a *silent* or *perfect* action, so called because it describes the *synchronization* of PickUp and $\overline{\mathit{PickUp}}$ that is entirely *internal* to the parallel process Env $\parallel$ Nav [14, 24].

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.

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

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

in which $\|$ is definable in terms of $+$. However, the key idea is the *expansion theorem* [14, 24]:

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

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

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

It can readily be seen that this theorem gives a reduction of the meaning of parallel composition to that of external choice.[10]

3.5 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:

$$\begin{aligned} \textsc{Nav} \;=\; & ((\mathit{Move} \cdot \textsc{Nav} + \mathit{PickUp} \cdot \textsc{Nav}) \\ & \parallel \textsc{Update}) \\ & \lhd (\mathit{Alarm} \cdot \textsc{EmergencyExit}) \end{aligned}$$

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

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

The theory of interrupt combinators can be found in [24].

[10] This relies upon the representability of finite processes in a *standard form* [14, 24].

3.6 Sequential composition

The execution monitoring we saw in the last section has much in common with the ideas of so-called "reactive planning". We saw in § 3.2.5 that our algebra of plans is adequate to describe the traditional plans produced by classical planning systems; we now see that it can also be used to describe reactive plans.

Consider, for example, Agre and Chapman's Pengi [2], 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

$$\begin{aligned} \textsc{Pengi} \;=\; & \mu_X(((\mathit{Cond}_1 \cdot \mathit{Action}_1 + \\ & \ldots + \mathit{Cond}_n \cdot \mathit{Action}_n) \\ & \parallel \textsc{Arbitration}) \,;\, X), \end{aligned}$$

where the ; combinator denotes the *sequential composition* of processes.[11] 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 [24]. Informally, the Q in P ; Q can proceed as soon as P has terminated.

4 Discussion

The use of the algebraic theory of processes to represent plans is not new. For example, both the Procedural Reasoning System (PRS) of Georgeff and Lansky [11, 10] and Lyons and Hendriks's $\mathcal{RS}$ [18, 19] are based on process theory. Lansky [16] introduces the notion of *action-based planning* for use in logistical planning domains. Firby's RAPS system [8, 9] and McDermott's RPL [22] are both essentially based on actions. However,

[11] 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.

little attention has been paid to the advantages of using the established results of process theory.

In this paper we have presented a simple representation of plans designed to facilitate their execution. We have based this representation on the algebraic theory of processes, and have demonstrated that it is able to represent a wide variety of the plans that appear in the AI planning literature. It can handle a number of important issues that arise in plan-execution, such as combining several plans and representing interruptions to actions as they are performed. We have argued that a representation based on processes has many advantages for the execution of plans over one based on state-changes.

We have not, however, 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 [29]. We believe that the large body of theoretical results provided by the use of a process algebra such as the one we have described will provide a sound basis for the necessary analysis.

There are two types of reasoning about actions and plans that must be considered: firstly, as designers of agents, we must be able to reason about the agents we design and their interactions with their environments; secondly, 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.

Initial investigations indicate that *Hennessy-Milner* (HM) logic [24] will prove to be a productive framework within which to reason about the behaviour of the processes that model our agent. HM logic provides us with a logical notion of equivalence of processes which turns out to be the same as a purely behavioural notion based on *bisimulation* [24]. This is one of the first results in the rich metatheory of processes, which also provides a natural way of comparing nondeterministic plans. Our theory of plan-comparison, which will be discussed elsewhere, is based on reasoning about the sets of possible plan outcomes. It includes but is not limited to decision-theoretic techniques. Moreover, the natural equivalence induced by this theory, *outcome-equivalence*, is compatible with an HM-like equivalence.

HM logic also allows us to reason about how processes affect propositions: it will thus allow us to reason about the effects of actions and the possibility of goal-achievement. Of particular importance in this respect is the satisfaction relation, $P \models \phi$, between processes P and (modal) propositions ϕ. This relation is defined inductively in the usual way and can be read as "after P has executed, ϕ holds" [24]. Suppose an agent's goal is expressed as a proposition ϕ. We may ask if, starting from the executing agent's repertoire of capabilities, we can construct, using the plan-combinators, a plan P such that $P \models \phi$. In this sense, plan-construction can be characterized as (model-based) theorem proving in HM logic.

Although HM logic allows us to describe and reason about possible behaviours in an elegant and intuitive way, it is not necessarily appropriate as the basis of the agent's own reasoning. In order to be able to reason about its own behaviour, and hence construct and adapt plans, an agent needs a suitably *intensional* logic that will allow it to derive processes that will achieve goals. Designing such logics (there will be choices, depending on our philosophical and engineering stances) is a non-trivial task and is a major area of future work.

References

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

[2] P. Agre and D. Chapman, *An implementation of a theory of activity*, Proceedings of the Sixth National Conference on Artificial Intelligence, 1987.

[3] D. Chapman, *Planning for Conjunctive Goals*, Artificial Intelligence Volume 32 (1987), pp. 333–337.

[4] G. Collins and L. Pryor, *Planning under uncertainty: Some key issues*, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, C. Mellish (editor), Morgan Kaufmann, 1995, pp. 1567–1573.

[5] D. Draper, S. Hanks and D. Weld, *Probabilistic Planning with Information Gathering and Contingent Execution*, Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL, AAAI Press, 1994, pp. 31–36.

[6] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson, *An approach to planning with incomplete information*, Proceedings of the Third International Conference on Knowledge Representation and Reasoning, Boston, MA, Morgan Kaufmann, 1992, pp. 115–125.

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

[8] R. J. Firby, *Adaptive execution in complex dynamic worlds*, Technical Report YALEU/CSD/RR 672, Computer Science Department, Yale University, 1989.

[9] R. J. Firby, *Task networks for controlling continuous processes*, Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL, AAAI Press, 1994, pp. 49–54.

[10] M. Georgeff and A. Lansky, *Procedural knowledge*, Proceedings of the Institute of Electrical and Electronics Engineers, Volume 74 (1986), Number 10, pp. 1383–1398.

[11] M. Georgeff, A. Lansky, and P. Bessiere, *A procedural logic*, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, 1985, pp. 517–523.

[12] R. van Glabbeek, B. Steffen and C. Tofts. *Reactive, generative and stratifed models of probabilistic processes*, Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS 90), Philadelphia, USA, IEEE Computer Society Press, Los Alamitos 1990, pp. 130–141.

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

[14] M. Hennessy, *An algebraic theory of processes*, M.I.T. Press, 1988.

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

[16] A. Lansky, *Action-based planning*, Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL, AAAI Press, 1994, pp. 110–115.

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

[18] D. Lyons, *Representing and analysing action plans as networks of concurrent processes*, IEEE Transactions on Robotics and Automation, Volume 9 (1993), Number 3, pp. 241–256.

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

[20] D. McAllester and D. Rosenblitt, *Systematic nonlinear planning*, Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA, AAAI Press, 1991, pp. 634–639.

[21] 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, 1969, pp. 463–502.

[22] D. McDermott, *A reactive plan language*, Technical Report YALEU/CSD/RR 864, Computer Science Department, Yale University, 1991.

[23] D. McDermott, *A Critique of Pure Reason*, Computational Intelligence, Volume 3 (1987), pp. 151–160.

[24] R. Milner, *Communication and concurrency*, International series on computer science, Prentice Hall International, 1989.

[25] E. P. D. Pednault, *ADL and the state-transition model of action*, Journal of Logic and Computation Volume 4 (1994), pp. 467–512.

[26] M. Peot and D. Smith, *Conditional Nonlinear Planning*, Proceedings of the First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland, Morgan Kaufmann, 1992, pp. 189–197.

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

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

[29] L. Pryor, *Opportunities and planning in an unpredictable world*, Ph.D. thesis, available as Technical Report No. 53, the Institute for the Learning Sciences, Northwestern University, 1994.

[30] L. Pryor, *Decisions, decisions: Knowledge goals in planning*, in *Hybrid problems, hybrid solutions* (Proceedings of AISB-95), J Hallam (Ed), pp. 181-192. IOS Press, 1995.

[31] L. Pryor and G. Collins, *Cassandra: Planning with contingencies*, Technical Report No. 41, the Institute for the Learning Sciences, Northwestern University, 1993.

[32] 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.

[33] D. Pym, L. Pryor and D. Murphy, *A note on processes for plan-execution and powerdomains for plan-comparison*, Available as Technical Report No. 719, Department of Computer Science, Queen Mary and Westfield College, University of London.