A conditional planning approach for the autonomous design of reactive and robust sequential control programs*

L. Castillo, J. Fdez-Olivares, A. González
Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S. Ingeniería Informática. Universidad de Granada
18071 Granada. {L.Castillo,faro,A.Gonzalez}@decsai.ugr.es

Abstract

In this work we present a conditional planning approach based on new semantical notions which allow the generation of correct conditional plans in real and uncertain domains. The planning algorithm which embodies these original semantics is based on POP techniques and it has a clear practical application: it can be seen as an autonomous design process able to obtain conditional plans which can be interpreted as *closed-loop* control programs.

Introduction

It is well known that current work on planning under uncertainty is mainly focused on the establishment of sound planning models (Bonet & Geffner 2000; Cimatti & Roveri 1999; Geffner 1998; Rintanen 1999; Son & Baral 2001) and fast planning algorithms (Weld, Anderson, & Smith 1998), however it is also known that they lack of real applicability (Wilkins 2001). Thus, a main conclusion of the last planning conference was concerned with the necessity of developing new approaches of planning under uncertainty with practical application.

This call for practical planning approaches, in the concrete field of conditional planning, involves at least three foundamental issues: (1) the development of planners which deal with actions that do not always have certain outcomes, and which assume that the state of the world will not always be completely known, (2) the development of planners which embody a model of actions expressive enough for real applications, and (3) the development of semantical concepts in order to support a planning algorithm which guarantees the quality of the results.

In this sense, this paper tackles the problem of practical planning in real and uncertain domains where several agents exhibit a behavior which is affected by the existence of sources of uncertainty, that is, logical or physical entities which supply information about the environment. Concretely, we are interested in the field of software engineering of sequential control programs for manufacturing systems, one of the main topics in which planning community is interested.

*This work has been supported by the spanish government CICYT under project TAP99-0535-C02-01.

A manufacturing system is composed of a set of devices which may be seen as a set of agents acting in an uncertain environment, whose behavior is conditioned by sensors, and which must be globally coordinated in order to achieve a common goal. That goal is a specification of a process on products which has to be carried out by the agents, which are coordinated by a *sequential control program* that guides the operation of the system. Figure 1 shows an example of a manufacturing system where we can find the following agents:

- A pump, P1, which transports water from the tank T1 to the tank T2, and which requires that the valve V1 be open. Additionally, there is a sensor, S_{level}, used to inform about the level of water at T1. The sensor can be in two possible states: on (T1 contains water) and off (T1 is empty). The behavior of P1 is restricted by the sensor S_{level}: when S_{level} is in the state on, P1 turns on, and when S_{level} is in the state off, it turns off.
- A pump, P3, which transports chlorine from the tank TC to T2, and which requires that the valve V3 be open.
- Two pumps, P21 and P22, which transport soda from the tank TS to T2. Both pumps require that the valve V2 be open, but only one of them may be active. The sensor S_{avble} can be in two states (s_1 or s_2),and it is used to decide what pump must be turned on.

The goal of this system is to obtain neutral pure water in T2 (initially contained in T1). Additionally, the sensor S_{pH} (which can be in three possible states $\{n, a, b\}$) informs whether the pH of the water contained in T2 is neutral (S_{pH} is in state n), acidic (S_{pH} is in state a) or basic (S_{pH} is in state b). Thus, the operation of the system must take into account that if the pH of the water contained in T2 is acidic then it will be necessary to add soda to T2, and if the pH is basic then it will necessary to add chlorine.

In order to obtain a control program for this manufacturing system, an expert follows a design process which receives as input the description of the system and a specification of its operation. The result of this process is a *closed-loop* control program, that is, a sequence of actions to be performed by the agents of the system, which take into account the information supplied by sensors, and which must incorporate conditional structures (possibly nested) in order to adequately describe the complex operation of the agents

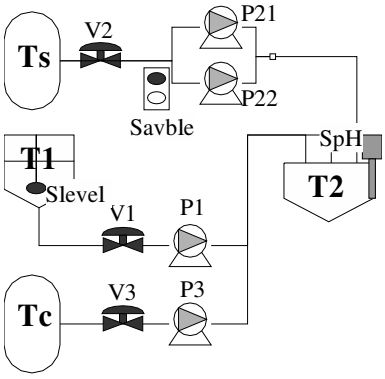

Figure 1: A manufacturing system for the neutralization of pure water.

of the system.

In this sense, the design process of a reactive and robust sequential control program is a problem that may be tackled within a planning with incomplete knowledge framework, taking into account that the application of planning techinques will require to obtain a complete solution by incorporating all the foreesable contingencies at running time. However, we have to rule out some possible alternative approaches as, for example, interleaving planning an execution (Koenig & Simmons 1998). The reason is that, in the context of manufacturing systems, the use of an execution process to improve a previously obtained incomplete plan might be extremely harmful for the system operation. Therefore, the best choose is a pure conditional planning approach.

In this context, previous work on planning applied to manufacturing operation (Aylett *et al.* 1998; PLANET 2001; Castillo, Fdez-Olivares, & González 2001a; 2001b) has shown that classical partial order planning may be applied as a successful technique for the autonomous design of *open-loop* control sequences. However none of these approaches assumes that the knowledge managed by a planner in a real domain may be incomplete. This is a shortcoming that limits the expressiveness of these approaches (that is, many real problems cannot be represented), and reduces the quality of their results (that is, the plans obtained cannot be conceived as "real" control programs because they lack of conditional structures).

Thus, the rest of this paper will be devoted to introduce a conditional planning approach, based on POP techniques, for real-world problem solving, with application to the autonomous design of closed-loop sequential control programs for manufacturing systems, and taking into account the following needs:

- The agents of a domain must be able to rapidly react to detected changes produced in the environment (For example, the pump P1 must be turned off when the sensor S_{level} has been detected to be in the state off).
- Agents must show a robust behavior, that is, they have to reach the proposed goal no matter what contingencies might be detected. (For example, the pH of the water must be neutral, independently of the state which the sensor S_{pH} is in).
- Plans obtained must incorporate actions capable of obtaining information from the environment, and actions capable of making decisions in run-time. In addition, plans must include (possibly nested) conditional structures in order to be accepted by human experts.

The paper is organized as follows: first we will introduce the knowledge representation (for domains, conditional plans and problems) used in this approach, next we will define a semantics for conditional plans and, finally, we will describe a conditional planning algorithm based in this semantics able to obtain correct conditional plans.

Knowledge Representation

In this section we will firstly describe how to represent and manage the incomplete knowledge originated by the existence of discrete sources of uncertainty in a domain that, in our application example, corresponds to the layout of a manufacturing system. Next we will introduce the knowledge representation used to describe domains, actions, problems, and plans.

Representing Incompleteness

In our approach we represent a source of uncertainty by means of a *sensor*. A *sensor* σ represents a discrete source of uncertainty, and it is associated with a finite set of *possible states* which are represented as symbols ($\mathcal{D}(\sigma) = \{u_1, \ldots, u_n\}$). In real domains, sensors affect the behavior of agents and, in the field of sequential control programming, this influence is represented by means of discrete variables which can take different values at run-time. During the design step of a control program, these variables are used by experts to describe the reactive or conditional behavior that, at running time, will be exhibited by the agents in a manufacturing system. This kind of variables may be represented in a planning model as an special type of planning variables called *run-time variables*, which have been used as a way for dealing with incompleteness in previous approaches of planning under uncertainty (Etzioni *et al.* 1992; Olawsky & Gini 1990; Olawsky, Krebsbach, & Gini 1995).

As other approaches do, we use run-time variables to represent that a source of uncertainty may affect the knowledge of the planner, although in a slightly different way. Concretely, a run-time variable *!x* is a variable which can be instantiated with constant symbols (belonging to a finite and discrete domain $\mathcal{D}(!x)$), and which is associated with a single sensor $Sensor(!x)$, in such a way that every possible value of *!x* is associated with a single possible state of σ.

The utility of a run-time variable is closely related with its use in the knowledge representation based on *literals*. In the planning model we are introducing, a literal l is represented as a tuple $(Atom(l)\ .\ Rc(l))$, where $Atom(l)$ is a predicate (called the atom of l) which may include run-time variables in its terms , and $Rc(l)$ is a set of *knowledge restrictions*. A knowledge restriction is represented as a special literal $(\text{KNOWN}\ \ \sigma\ u)$, where σ stands for a sensor and u is one of its possible states. This extended syntax involves

some issues about the interpretation of a literal that we have to clarify:

- First, the truth-value of a literal l is the truth-value of *Atom*(l), which may be $true$, $false$ or $unknown$.
- Second, the value that a run-time variable *!x* may take (at running time) is unknown at planning time, and it always depends on a possible state of *Sensor*$(!x)$. Therefore, for a given literal l, if *Atom*(l) contains a run-time variable *!x*, then the truth-value of a complete instantiation of *Atom*(l) is restricted by the sensor *Sensor*$(!x)$, in such a way that it depends on a *knowledge production* caused by the detection of a state of *Sensor*$(!x)$.
- Third, taking into account the previous point, the set of knowledge restrictions of a literal allows to represent, at planning time, what sources of uncertainty (sensors) affect its truth-value, that is, knowledge restrictions are used to represent a *context* where a literal is known to be true or false. For example $l = (p\ .\ ((\text{KNOWN}\ \sigma\ u)(\text{KNOWN}\ \sigma'\ u')))$ is interpreted as "l is true when it has been detected that, simultaneously, σ is in the possible state u, and σ' is in the possible state u'", or in other words, "l is true in a context where σ takes the state u and σ' takes the state u'". Additionally, for a given literal l, *Rc*(l) may take the value T, meaning that there is no sensor which restricts the truth-value of l[1] , or the value NIL, meaning that the truth-value of l cannot be determined, representing an inconsistency.
- Finally, we have to say that not every set of restrictions is a valid one. Concretely, T is a valid set of knowledge restrictions, NIL is not a valid set, and a set of knowledge restrictions which contains two restrictions which refer to the same sensor is not a valid set (for example, the set $((\text{KNOWN}\ \sigma\ u_1)(\text{KNOWN}\ \sigma\ u_2))$ is not a valid one). Finally, a set of knowledge restrictions is valid when none of these rules applies.

The representation of knowledge restrictions is inspired on the concepts of *conditional context* and *context labels* used in (Peot & Smith 1992) and (Pryor & Collins 1996), and the validity of a set of knowledge restrictions is similar to the notion of *contexts compatibility* introduced in (Peot & Smith 1992). However, the model of actions of these known conditional planning approaches is basicaly the classical model of STRIPS, extended with sensing (or observe) actions. So, though these approaches manage uncertainty, their action representation is not expressive enough to face with problems in manufacturing domains. What is more, these models lack of a clear semantics that justify their planning algorithms.

Thus, the representation of knowledge restrictions will allow the introduction of new semantical concepts which will be the basis of a new conditional planning algorithm based on POP techniques, able to obtain ready-to-use sequential control programs. These new concepts will be based on an unification algorithm for literals with knowledge restrictions, which extends the classical unification algorithm, and which we describe next.

Definition 1 *Two literals, l_e and l, unify when their atoms unify and the set union of their knowledge restrictions is valid. In this case, the resulting literal l_u = **Unify**(l_e, l) is a literal defined as follows:*

Atom(l_u) *is the result of the classical unification of Atom*(l_e) *and Atom*(l)*, extended for run-time variables (a run-time variable may unify with a constant, or with a normal variable which contains a single constant in its codesignation constraints).*

Rc(l_u) *is a set of knowledge restrictions which includes Rc*(l_e)*, Rc*(l)*, and a newly generated set r_u of knowledge restrictions. Every knowledge restriction* $(\text{KNOWN}\ \sigma\ u)$ *of r_u is generated as result of the unification of a run-time variable !x, such that Sensor*$(!x) = \sigma$*, with a constant $k \in \mathcal{D}(!x)$, such that u is a state of σ associated with the value k.* □

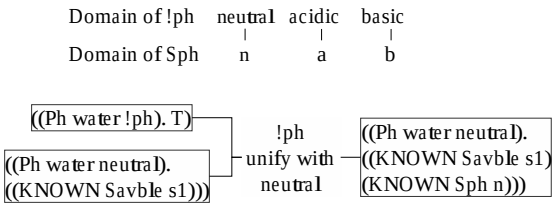

Figure 2: Unification of literals with knowledge restrictions.

Figure 2 shows an example of unification and, as can be seen, when a run-time variable is instantiated by a constant, the unification may lead to add new knowledge restrictions in the resulting literal. This means that, in order to determine the truth-value of the resulting literal, it is required to produce more knowledge than the one required to determine the truth-value of the unified literals.

Domain and Plans representation

In our model a domain is represented as a set of *agents* and a set of *sensors*. Every sensor σ is associated with a set of *sensing actions*. A sensing action is associated with a single state u of a sensor σ, and it is represented with a name (denoted as ***When***$(\sigma\ u)$) and with a set of effects. The effects of a sensing action ***When***$(\sigma\ u)$ represent facts of the world which are changed when a sensor σ is in the state u (See Figure 3). So, sensing actions are used to have access to these facts.

On the other hand, every agent g is represented by means of its properties (name, normal variables and run-time variables) and its behavior. The behavior is modeled as a finite automaton in which every *causal action* a which is executed by g contains a set *Req*(a) of literals, called *requirements*, which must be solved in order to achieve a correct execution of the action, and a set of effects, *Efs*(a), which include the change of state produced by a over the agent g. In order to achieve an adequate expressiveness for real-world planning,

[1]For operational purposes, we assume that T is included in any set of restrictions

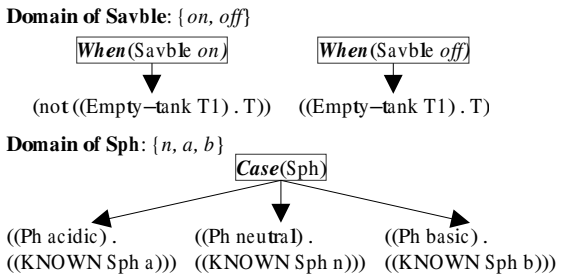

Figure 3: Sensing and decide-actions.

the representation of causal actions embodies the following features:

- Actions are considered as intervals, that is, every causal action a of an agent g executes over an interval, $[a, \mathit{End}(a)]$, defined from a until the next change of state of g, produced by another action, $\mathit{End}(a)$, of g.
- There are four kind of requirements: *previous* (they must be true before the action), *simultaneous* (they must be true during the interval of an action), *query* (used to find out facts before the action), and procedural (they must be true after the action) (See (Castillo, Fdez-Olivares, & González 2000; 2001a) for more details). Previous, simultaneous, and procedural requirements can only be solved by causal actions, but query requirements may be solved by causal or sensing actions.
- Every action a contains a set of knowledge restrictions, $\mathit{Rc}(a)$, which represent the context which a can be executed in (that is, the execution of an action may be affected by some states of the sensors in the domain).
- The effects of a causal action may contain literals with run-time variables. This means that, as a literal with run-time variables cannot be known at planning time, every action which contains run-time variables in its effects is a non-deterministic action. Additionally, in this work we assume, for simplicity purposes, that requirements do not contain run-time variables, which only one literal with run-time variables is allowed in the effects of a causal action, and which this literal can only contain one run-time variable.

The action model also includes another kind of actions called *decide-actions*. A decide action is associated with a single sensor σ, and it is represented with a name (denoted as $\boldsymbol{Case}(\sigma)$) and with a set of *possible effects* $\mathit{Efs}(\boldsymbol{Case}(\sigma)) = \{\mathit{PEf}_{u_1}, \ldots, \mathit{PEf}_{u_n}\}$ ($\{u_1, \ldots, u_n\}$ are the possible states of σ). Every possible effect PEf_{u_i} is a set of literals with knowledge restrictions, representing a causal transformation of the world in case of σ is in the possible state u_i (See Figure 3). Decide-actions are used at running time to make decisions about which course of actions must be followed, and, at planning time, they are automatically generated to represent the different outcomes of the execution of a non-deterministic action. This process will be described later, next we will describe how problems and plans are represented.

Definition 2 *A problem is represented as a tuple $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$ where $\mathcal{D}$ is a domain,$\mathcal{O}$ is a set of (possibly ordered) literals which represent the high level goal, and $\mathcal{I}$ is a set of literals which represent an incomplete initial state. Every literal l in the initial state can be initially true, false or unknown, in such a way that:*

- *l is initially true if $l \in \mathcal{I}$ and l does not contain run-time variables.*
- *l is initially unknown if some of the following conditions holds:*
  - *i) l or $(not\ l)$ belongs to the effects of some sensing-action of the domain.*
  - *ii) l or $(not\ l) \in \mathcal{I}$ and it contains run-time variables.*
- *l is initially false if some of the following conditions holds:*
  - *i) $(not\ l) \in \mathcal{I}$ and l does not contain run-time variables.*
  - *ii) $l \notin \mathcal{I}$ and l is not initially unknown.* □

Definition 3 *A conditional plan is represented as a tuple $\Pi = \langle A_c, A_{when}, A_{case}, < \rangle$ where $A_c(\Pi), A_{when}(\Pi), A_{case}(\Pi)$ stand for a set of causal, sensing and decide actions, respectively, and $<$ stands for a partial order relation between them. A conditional plan contains two dummy causal actions: a_0, the first action of a conditional plan, which is a non-deterministic action whose effects encode the incomplete initial state[2], and a_∞, the last action of any conditional plan and whose requirements encode the goal.* □

Semantics

This section is centered on the study of which conditions have to be accomplished for a conditional plan to be correct. Intuitively, a conditional plan it said to be correct if every action in the plan is executable, the execution of the plan allows to reach the goal of the problem, and there are no conflicts between the actions of the plan.

Firstly, we have to say that sensing actions do not contain requirements because their execution depends on a sensor to be in a possible state, which is a non-foreseeable event. However, when a sensing action $\boldsymbol{When}(\sigma\ u)$ executes, its effects are known to be true, and it is also known that the sensor σ is in the state u. Thus, sensing actions will be used to satisfy knowledge needs, which will be represented in the query requirements of causal actions.

On the other hand the execution of a decide-action $\boldsymbol{Case}(\sigma)$ can be interpreted has the simultaneous execution of all the sensing actions associated with σ, which means that the uncertainty about the current state of σ is eliminated after a decide-action. The utility of decide-actions will be detailed later.

Thus, in this approach the executability conditions of sensing and decide actions are not a subject to be studied, so, next section will be centered on the study of the executability conditions of a causal action in a conditional plan, and how can we interpret the execution of deterministic and

[2]It is allowed for the effects of the special action a_0 to contain more than one literal with run-time variables

non-deterministic actions. Afterwards, we will introduce the semantical concepts which will allow to accomplish these conditions by means of literal satisfaction, and we will describe the causal structure of a conditional plan. This will finally lead to a definition of correct conditional plan.

Executability conditions

Definition 4 *We will say that a causal action a, such that $Rc(a) = r$, is executable when the following conditions hold:*

i) Their requirements are true.

ii) $\forall l \in Req(a), Rc(l) = r \vee Rc(l) = \mathrm{T}$. □

This definition establishes that every requirement of an executable action may either be true in the same context in which the action can be executed, or be true in all possible contexts (that is, when its set of knowledge restrictions is equal to T). Additionally, we have to define what we consider to be a correct execution of an action.

Definition 5 *We will say that the execution of an executable and deterministic action is correct when* $\forall l_e \in Efs(a), Rc(l_e) = Rc(a)$. □

That is, if a deterministic action executes in a context $Rc(a)$, then its effects are true in the same context.

However, this definition does not apply for non-deterministic actions. The effects of this kind of actions contain run-time variables, meaning that some literals of the effects of a non-deterministic action will be true in different contexts. Therefore, the truth-value of these literals can only be determined at running time but, at planning time, we can make use of decide-actions in order to represent the possible transformations produced by the execution of a non-deterministic action. This will be done by associating decide-actions to non-deterministic actions, in such a way that the effects of a decide-action will be used to represent the different outcomes of a non-deterministic action.

In this sense, for every literal in the effects of a non-deterministic action a, containing a run-time variable *!x* such that $Sensor(!x) = \sigma$, will be associated a decide-action $\textbf{\textit{Case}}(\sigma)$ to a (See Figure 4). This leads to define a process which allows to automatically generate decide-actions within a conditional plan.

Definition 6 *Let a be a non-deterministic action, such that $Efs(a)$ contains a literal l with a run-time variable !x. When a is included in a conditional plan, a decide-action is generated in that plan according to the following function:*

$\textbf{\textit{GenDec}}(a, !x)$
Let l be the literal of $Efs(a)$ which contains !x
Let $\sigma = Sensor(!x)$
FOR EACH $k_i \in \mathcal{D}(!x)$
  Let u_i be the associated state of σ with k_i
  Let p_i represent the atom of l where !x is unified with k_i
  Let $PEf_{u_i}(\textbf{\textit{Case}}(\sigma)) = \{(p_i \,.\, Rc(l) \cup ((\textsc{known}\ \sigma\ u_i)))\}$
Let $Efs(\textbf{\textit{Case}}(\sigma)) = \bigcup_{u_i} PEf_{u_i}(\textbf{\textit{Case}}(\sigma))$
RETUN $\textbf{\textit{Case}}(\sigma)$

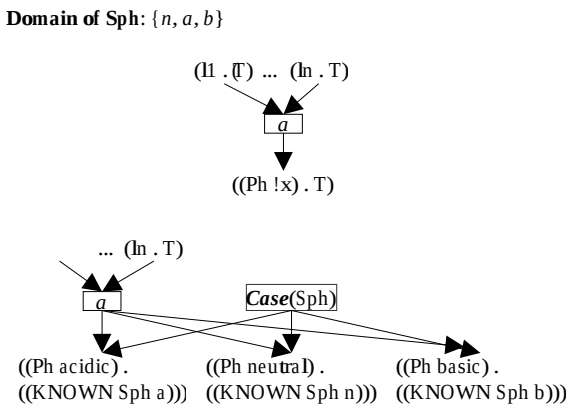

Figure 4: A decide action associated to a causal action.

Additionally, the possible effects of $\textbf{\textit{GenDec}}(a, !x)$ *become effects of a.* □

The generation of decide-actions will support the representation, in a single conditional plan, of different sequences of actions executed in different contexts, which can be seen as conditional branches. However, the existence of a decide-action in a conditional plan is not sufficient to create different conditional branches. The creation of conditional branches will be discussed in the next section, where we will introduce how to accomplish the executability conditions of causal actions (deterministic or not) when different execution contexts are represented in a conditional plan.

Different modes of satisfaction

In classical POP, a literal l in the requirements of an action a, included in a plan, may become true if there is another previous action b which satisfies l, that is, if this action contains a literal l_e in its effects which unifies with l. The process followed by our approach to determine the truth-value of the requirements of an action is also based on literal satisfaction. However, the new features of the unification algorithm described above lead to distinguish between different modes of satisfaction (See Figure 5), taking into account whether the unification generates new knowledge restrictions or not.

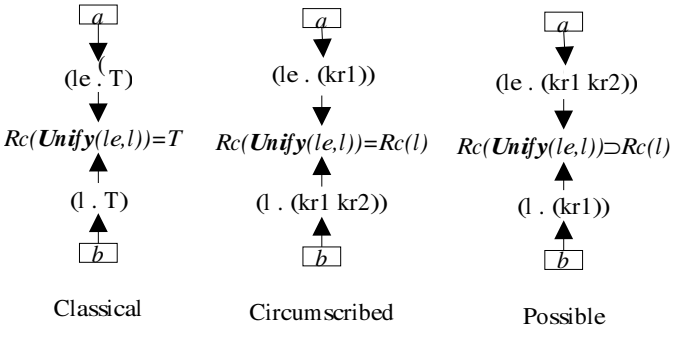

Figure 5: Different modes of satisfaction.

Definition 7 *Let b be a causal action with a literal l in its requirements, and let a be an action (of any kind) which contains a literal l_e in its effects which unifies with l, such that $a < b$ (if the requirement is procedural we consider $b < a$). The unification of l_e and l may lead to the following cases:*

- *Rc(**Unify**$(l_e, l)) = \mathrm{T}$: this is the case of classical satisfaction (noted as $a \xrightarrow{Sat} l$).*
- *Rc(**Unify**$(l_e, l)) \neq \mathrm{T}$ and Rc(**Unify**$(l_e, l)) =$ Rc(l): this case will be referred to as circumscribed satisfaction (noted as $a \xrightarrow{\mathbf{CSat}} l$).*
- *Rc(**Unify**$(l_e, l) \supset$ Rc(l): this case will be referred to as possible satisfaction supplying a set of knowledge restrictions defined as $r_a =$ Rc(**Unify**$(l_e, l) -$ Rc(l). (noted as $a \xrightarrow{\mathbf{PSat}} [l, r_a]$).* □

In the following, we will describe how these modes of satisfaction affect to the accomplishment of the executability conditions of a causal action.

When $a \xrightarrow{\mathbf{CSat}} l$ holds, for some action a and some literal l of the requirements of an action b, we have to take into account two possible cases: Rc$(a) =$ Rc(l) or Rc$(a) \in$ Rc(l). The first case is consistent with Definitions 4 and 5, but the second one violates the definition of correct execution for the action a , because b requires a to be executable in the context Rc(l), and a is known to be executable in a different context. So, in order to guarantee the accomplishment of Definition 5, it will be necessary to *propagate backwards* (backto the action a, its requirements and effects) a set of knowledge restrictions, following a process defined as follows:

Definition 8 *Let a and b be two causal actions such that $a \xrightarrow{\mathbf{CSat}} l$ holds, for some requirement l of b. The backward propagation of knowledge restrictions from b to a is defined by the following rule:*

BckProp(a, l)
IF Rc$(a) \subset$ Rc(l)
THEN FOR EACH $l \in$ Req$(a) \cup$ Efs(a)
ASSIGN Rc$(l) =$ Rc$(l) \cup$ Rc$(b) -$ Rc(a)
ASSIGN Rc$(a) =$ Rc$(a) \cup$ (Rc$(b) -$ Rc(a))
RETURN a

□

With respect to the case of a possible satisfaction, it is important to note that when $a \xrightarrow{\mathbf{PSat}} [l, r_a]$ holds, for some action a and for some literal l of the requirements of an action b, the Definition 4 is violated (because a requirement of b would be true in a context different from Rc(b)). In this case, in order to guarantee the executability conditions of b, there are two alternatives: to assume that l can either be true in the context Rc$(l) \cup r_a$, or be necessarily true in the context Rc(l).

In the first case, the executability conditions of b can be accomplished by means of a process which *propagate forwards* (toward the action b, its requirements and effects) the set of restrictions r_a. However, this propagation cannot be applied in any case. Concretely, if the literal satisfied is contained in the main goal, or b is a non-deterministic action whose effects contain a literal l_j, such that Rc$(l_j) \cup r_a$ is not valid, then forward propagation does not apply. This is defined in the following rule.

Definition 9 *Let a and b be two causal actions such that $a \xrightarrow{\mathbf{PSat}} [l, r_a]$ holds, for some requirement l of b. The forward propagation of the knowledge restrictions set r_a, from a towards b, is defined by the following rule:*

FwProp(a, l, b, r_a)
IF l *is a goal literal or*
b *is a non-deterministic action such that*
$\exists l_k \in$ Efs(b), Rc$(l_k) \cup r_a$ *is not valid*
THEN RETURN *FAIL*
ELSE FOR EACH$l \in$ Req$(b) \cup$ Efs(b)
ASSIGN Rc$(l) =$ Rc$(l) \cup r_a$
ASSIGN Rc$(b) =$ Rc$(b) \cup r_a$
RETURN a

□

It is necessary to remark that, indeed, forward propagation can be recursively applied through the causal structure of a conditional plan. This recursive process will be described in the next section.

In the case of $a \xrightarrow{\mathbf{PSat}} [l, r_a]$ holds, for a literal $l = (p\ .\ r)$, and forward propagation cannot be applied, it is interpreted that l must be necessarily true in the context r. This leads to define a new mode of literal satisfaction which will be referred to as *necessary satisfaction*. This concept will be defined just after we have described how to represent that a literal l must be necessarily true in a context r when $a \xrightarrow{\mathbf{PSat}} [l, r_a]$ holds (See Figure 6), by means of the concept of *conditional expansion*.

Definition 10 *Let a and b be two causal actions such that $a \xrightarrow{\mathbf{PSat}} [l, r_a]$ holds, for some literal l of the requirements of b. The conditional expansion of l with respect to the knowledge restrictions set r_a, is a set of literals defined by de following function:*

C-exp(l, r_a)
IF $r_a = \{\emptyset\}$ THEN RETURN $\{l\}$
IF $r_a = ((\mathrm{KNOWN}\ \sigma\ u)))$
THEN *Let* $C = \{\emptyset\}$
FOR EACH $u_i \in \mathcal{D}(\sigma)$
IF Rc$(l) \cup ((\mathrm{KNOWN}\ \sigma\ u_i))$ *is valid*
THEN *Let* $C = C \cup \{($Atom$(l)\ .$ Rc$(l) \cup$
$((\mathrm{KNOWN}\ \sigma\ u_i)))\}$
RETURN C
ELSE *Let* $r = (\mathrm{KNOWN}\ \sigma\ u))$ *(extracted from* r_a*)*
Let $C = \{\emptyset\}$
FOR EACH $u_i \in E(\sigma)$
IF Rc$(l) \cup ((\mathrm{KNOWN}\ \sigma\ u_i))$ *is valid*
THEN *Let* $C = C \cup$ ***C-exp***$(($Atom$(l)\ .$ Rc$(l) \cup$
$((\mathrm{KNOWN}\ \sigma\ u_i))), r_a - r)$
RETURN C

□

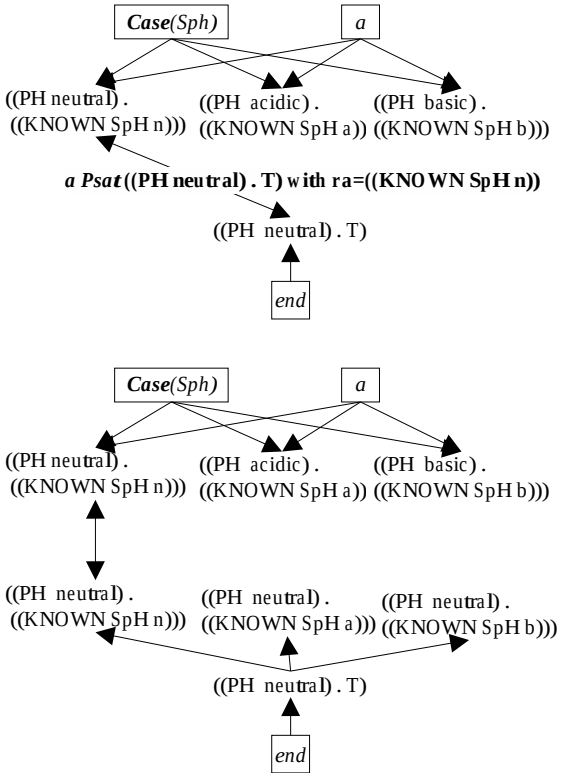

Figure 6: Conditional expansion of a literal with respect to r_a.

That is, the conditional expansion of a literal l, with respect to r_a is a set of literals that allow to represent that l must be true in the context $Rc(l) \cup r_a$, and in any other possible context $Rc(l) \cup r_i$, where r_i stands for a valid combination of the possible states of the sensors included in r_a (See Figure 6). The conditional expansion of a literal is the basis of the concept of necessary satisfaction which is defined as follows:

Definition 11 *Let a and b be two causal actions such that $a \xrightarrow{PSat} [l, r_a]$ holds for some literal in the requirements of b. We will say which l is necessarily satisfied in a conditional plan Π (noted as $\Pi \xrightarrow{NSat} [l, a]$) when $\forall l_c \in$ **C-exp**(l, r_a), $\exists a_c \in A_c(\Pi)$, such that $a_c \xrightarrow{CSat} l_c$ holds.* □

That is, a literal is necessarily satisfied in a conditional plan when the literals of its conditional expansion are satisfied in a circumscribed mode. In addition, it is worth noting that this mode of satisfaction will allow to generate, in a conditional plan, by means of the regression of the literals of a conditional expansion, sequences of actions which will satisfy (in a circumscribed mode) every literal of **C-exp**(l, a), for some literal l and some action a which satisfies l in a possible mode. The actions of these sequences will be executed in different contexts and, so, we can conclude that the conditional expansion of a literal l will allow to create conditional branches within a conditional plan.

Finally, we can redefine the conditions under a causal action is considered to be executable, in terms of the modes of satisfaction previously introduced.

Definition 12 *A causal action is executable if its previous, simultaneous, and procedural requirements are satisfied in a circumscribed or necessary mode, and its query requirements are satisfied in a circumscribed mode.*

Different kinds of causal links

In the previous section we have shown that the requirements of a causal action may be satisfied in different modes by different kinds of actions. This means that, in addition to causal links, the causal structure of a conditional plan embodies different kinds of causal dependencies. These kinds are the following ones:

- *Causal link*, noted as $[a \xrightarrow{l} b, c]$, which represents that a requirement l of an action b has been satisfied in a circumscribed or classical mode by a causal action a, and which has to be protected during the actions interval $[a, c]$ (if l is a previous or query requirement $c = b$, and if l is simultaneous $c = End(b)$).

- *Detection link*, noted as $\mathbf{D}[a \xrightarrow{l} b]$, which represents that a query requirement l of b has been satisfied in a circumscribed or classical mode by a sensing action a, and which has to be protected during the actions interval $[a, b]$.

- *Possible link*, noted as $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$. which represents that literal l_n which belongs to the conditional expansion of a previous, simultaneous, or procedural requirement l_n of an action b, has been satisfied in a circumscribed mode, and which has to be protected during the actions interval $[a, c]$.

These kinds of causal dependencies are mainly used to detect threats between actions in a conditional plan. The threats management is similar to the one followed in classical POP, although it is necessary to note that it is extended to incorporate the notion of actions interval and the representation of literals here presented. In order to define the concept of threat, it is necessary to know that an action a' (of any kind) negates the literal l when there is a literal $l' \in Efs(a)$ such that $Atom(l')$ negates $Atom(l)$ and $Rc(l') \cup Rc(l)$ is valid.

Thus, we can define the following types of threats:

- A causal action a' threatens a causal link $[a \xrightarrow{l} b, c]$ (or a possible link $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$) when a' negates l and the interval $[a, c]$ is unordered with respect to the interval $[a', End(a')]$.

- A sensing action a' threatens a detection link $\mathbf{D}[a \xrightarrow{l} b]$ when a' negates l and the interval $[a, b]$ is unordered with respect to a'.

Detection links, and the threats management for these links, allow to introduce order constraints between sensing actions, meaning that it is possible to describe a correct reactive behaviour for the agents of a domain.

On the other hand, causal and possible links allow to support a recursive forward propagation of knowledge restrictions, based on the forward propagation rule above defined (Definition 9). This recursive process starts when $a \xrightarrow{\textbf{\textit{PSat}}} [l, r_a]$ holds for some action a and some requirement l of another action b, and r_a are propagated towards b.

In this case, the causal dependencies (causal or possible links) which are produced by b must be revised, because the context of b has changed, and it is possible that the executabilty conditions of every action supported by b would be violated. Therefore, in order to guarantee the correct execution of every action a_b supported by b, it will be necessary to test whether these conditions have to be updated or not, that is, whether r_a must be propagated towards a_b. This lead to a "test-and-propagate" recursive process, through the causal structure of a conditional plan, which ends when the supported action is a_∞ or r_a cannot be propagated forward. This process is described in the following definition.

Definition 13 *Let a and b be two causal actions such that $a \xrightarrow{\textbf{\textit{PSat}}} [l, r_a]$ holds, for some literal l in the requirements of b. Let cl a causal or possible link which represents a causal dependence between a and b. The recursive propagation of r_a, from vc and through the causal structure of a conditional plan, is defined by the following function:*

R-FwProp(vc, r_a)
Let a, b and l be theproducer action, consumer action and the literal of vc, respectively
IF vc *is not a possible link*
THEN IF $b = a_\infty$ THEN RETURN $\{vc\}$
ELSE IF ***FwProp***(a, l, b, r_a) *does not apply*
THEN IF $r_a \cup Rc(b)$ *is not valid*
THEN RETURN *FAIL*
ELSE RETURN $\{vc\}$
ELSE *Let $VC = \{vc\}$ be a set of links*
FOR EACH *link vc_b such that b is its producer*
stop-prop = ***FwProp***(vc_b, r_a)
IF *stop-prop = FAIL* RETURN *FAIL*
ELSE $VC = VC \cup$ *stop-prop*
RETURN VC
ELSE RETURN $\{vc\}$□

The function **R-FwProp**(vc, r_a) returns a set of causal or possible links which contains those causal dependencies where forward propagation cannot be applied. Thus, every link returned can be seen as the representation of a possible satisfaction where it is not possible to propagate forwards. If during the recursive propagation a not valid set of knowledge restrictions is found, the function will return a general fail, which means that the forward propagation through the causal structure will not lead to obtain a correct plan. This function will be very useful in the conditional planning algorithm wich will be introduced in the next section. However, previously we have to establish the conditions under a conditional plan is considered to be correct.

Definition 14 *A conditional plan* Π*, constructed to solve a planning problem $\mathcal{P} = \langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$ is correct when every causal action in* Π *is executable, every literal in $\mathcal{O}$ is satisfied in a circumscribed or necessary mode by any executable action of* Π*, and there are no threats between the actions intervals of* Π*.* □

SolveSubGoal(l, a, Π)
Let b be the action such that $l \in$ *Req*(b).
Let c be the action *End*(b), if l is a simultaneous requirement, or the action b, if l is a previous requirement
IF a is a new action
THEN Insert a in Π
IF a is a non-deterministic action
THEN Insert **GenDec**$(a, !x)$ in Π
CASE $a \xrightarrow{\textbf{\textit{CSat}}} l$
IF l is a query requirement and a is a sensing-action
THEN Insert $\mathbf{D}[a \xrightarrow{l} b]$
ELSEIF l belongs to the conditional expansion of some literal l_n
THEN Insert $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$ in Π and **BckProp**(a, l)
ELSE Insert $[a \xrightarrow{l} b, c]$ in Π and **BckProp**(a, l)
CASE $a \xrightarrow{\textbf{\textit{PSat}}} [l, r_a]$
IF l belongs to the conditional expansion of some literal l_n
THEN Insert $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$ in Π
stop-prop =$\{\mathbf{P}[a \xrightarrow{l} l_n, b, c]\}$
ELSE Insert $[a \xrightarrow{l} b, c]$ in Π
stop-prop = **R-FwProp**$([a \xrightarrow{l} b, c], r_a)$
IF stop-prop = FAIL RETURN FAIL
FOR EACH vc in stop-prop
Insert the refinement task "Conditional Branching vc, r_a"

Figure 7: Algorithm for solving an unsolved goal flaw.

In the next section we will describe an algorithm able to obtain correct conditional plans according to this definition.

Planning algortithm

The previously defined notions have been incorporated into the partial order algorithm described in (Castillo, Fdez-Olivares, & González 2001a; 2000) resulting in a new conditional planning algorithm (called ADVICE), capable of constructing correct conditional plans, in the terms defined above, which can be interpreted as *closed-loop* control programs. Thus, ADVICE is based on a planning algorithm which generates a conditional plan by the successive application of refinement operations on a partially constructed conditional plan. These operations are carried out when, in the partially constructed conditional plan, it appears any of the following flaws:

- *Threats*. ADVICE solves the different types of threats defined above by demotion or promotion of actions intervals (See (Castillo, Fdez-Olivares, & González 2001a) for more details).
- *Unsolved subgoals*. Subgoals are solved following the algorithm shown in Figure 7. A literal which represents an unsolved subgoal may be satisfied by a non instatiated causal action or by an action included in the conditional plan in construction (if the literal represents a query requirement, the action may be a sensing action). In the first

MakeBranch(vc, r_a, Π)
IF vc is a causal link $[a \xrightarrow{l} b, c]$
THEN Insert the literals of **C-exp**(l, r_a) as new subgoals in Π
IF vc is a possible link $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$
THEN Non deterministally chosse a literal l' from the set $\{l, l_n\}$
Insert the literals of **C-exp**(l', r_a) as new subgoals in Π

Figure 8: Algorithm for conditional branch creation.

case, ADVICE inserts the action, associating a decide-action in case of the action is non-deterministic. If the literal is satisfied in a circumscribed mode, a backward propagation of knowledge constraints is performed. Then, if the literal belongs to the conditional exansion of another requirement, a possible link is inserted, otherwise, a causal link is inserted (if the literal is a query requirement satisfied by a sensing action, a detection link is inserted). On the other hand, if the literal is satisfied in a possible mode (we assume that query requirements cannot be satisfied in a possible mode), causal and possible links are inserted as in the previous case , and a recursive forward propagation process is performed (we assume that knowledge restrictions cannot be propagated towards the literals of a conditional expansion). If this process returns a fail, it is interpreted that the subgoal cannot be solved by the action being used, otherwise, every causal or possible link returned by this process will be considered as a new flaw of type "Conditional-Branching".

- *Conditional branching*. This new type of flaw raises when the forward propagation process cannot be applied on a causal or possible link. We have introduced above that every link returned by this process represents a possible satisfaction where it is not possible to propagate forwards. Therefore, according to Definitions 10 and 11, this flaw can be solved by means of a conditional expansion (See Figure 8). Thus, the introduction of the literal of the a conditional expansion as unsolved subgoals, will allow to guarantee that a literal can be satisfied in a necessary mode by the creation of different conditional branches. In addition, as can be seen in Figure 8, it is not possible to determine, in a possible link $\mathbf{P}[a \xrightarrow{l} l_n, b, c]$, which literal must be conditionally expanded (l or l_n). In this case, the flaw can be solved by means of two alternatives.

Figure 9 shows the conditional plan generated by ADVICE which solves the manufacturing problem described in the introduction of this paper. The plan represents correctly the required behaviour:

- The pump P1 and the valve V1 *react* to the information supplied by the sensor S_{level}, in such a way that they turn off when S_{level} is in the state off.
- On the other hand, the actions of the agents V2, V3, P21, P22, and P3, are conditionally structured in such a way that these agents show a robust behavior which allows to obtain a neutral pH. In addition, the plan contains nested conditional branches (which may be discover by means of the recursive propagation process defined above) to represent the alternative operation of the agents P21 and P22.

As can be seen in this example, ADVICE satisfies the needs formerly established in the introduction of this work.

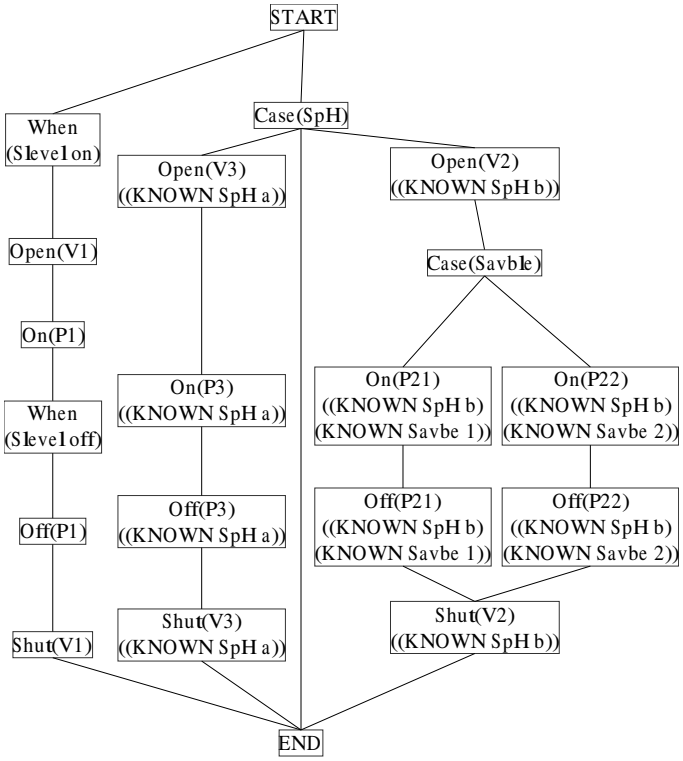

Figure 9: A conditional plan for the problem formerly introduced.

Conclusions

In this work we have presented a conditional planning approach based on an expressive model of actions, and on new semantical notions which allow to generate correct conditional plans, in real and uncertain domains. The planning algorithm which embodies these original semantics can be seen as an autonomous design process able to obtain conditional plans, which can be interpreted as ready-to-use *closed-loop* control programs.

Taking into account the planning process and the plans obtained, ADVICE can be seen as a step forward in the field of planning applied to software engineering, overwhelming the main shortcomings (concerned with incomplete knowledge management and plan quality) of current planning approaches to manufacturing systems operation. It must also be said that ADVICE has been extensively used for the automatic synthesis of closed-loop control programs. This experimentation is being carried out in close collaboration with experts on industrial domains within a research project, and it will appear in other paper in preparation.

On the other hand, appart from the semantical concepts presented, one of the main advantages of this approach is

that it embodies a more expressive model of actions than the one used in other conditional planning approaches, allowing to obtain conditional plans with nested conditional structures. That is, as opposite to other POP approaches which assume incomplete knowledge (Pryor & Collins 1996; Etzioni *et al.* 1992), the representation of a plan obtained by ADVICE is a DAG and not a tree (Weld, Anderson, & Smith 1998), a characteristic that needs to be incorporated into any plan intended to be useful and understandable for human experts.

Our current research is currently focused on the integration of this conditional planning approach with the hybrid planning model (which mixes hierarchical planning and POP, but which assumes complete knowledge) described in (Castillo, Fdez-Olivares, & González 2001b)). This will lead to develop a planning system able to obtain hierarchical and conditional plans which could be interpreted as hierarchical and conditional control programs. We think that such as system will be extremely helpful to the experts on industrial automation.

References

Aylett, R.; Soutter, J.; Petley, G.; and Chung, P. 1998. AI planning in a chemical plant domain. In *European conference on artificial intelligence*.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, 52–61.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2000. A three-level knowledge-based system for the generation of live and safe petri nets for manufacturing systems. *Journal of Intelligent Manufacturing* 11(6):559–572.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2001a. Mixing expresiveness and efficiency in a manufacturing planner. *Journal of Experimental & Theoretical Artificial Intelligence (JETAI)* 13:141–162.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2001b. On the Adequacy of Hierarchical planning characteristics for Real-World Problem Solving. In *Proceedings of the Sixth European Conference on Planning ECP-01)*.

Cimatti, A., and Roveri, M. 1999. Conformant planning via model checking. In *Proceedings of the Fifth European Conference on Planning (ECP-99)*, 21–33.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proc. Third. Int. Conf. on Principles of KRR-92*, 115–125.

Geffner, H. 1998. Modelling intelligent behaviour: The Markov decision process approach. *Lecture Notes in Computer Science* 1484:1–12.

Koenig, S., and Simmons, R. 1998. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning System*, 154–163.

Olawsky, D., and Gini, M. 1990. Deferred planning and sensor use. In *Innovative Approaches to Planning, Scheduling and Control*, 166–174. Morgan Kauffman, San Mateo.

Olawsky, D.; Krebsbach, K.; and Gini, M. 1995. An analysis of sensor-based task planning. Technical Report TR95-51, Dep. Comp. Science. University of Minneapolis.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In *Proc. First Int. Conf. of AIPS*, 189–197.

PLANET. 2001. The PLANET roadmap on AI planning and scheduling. http://www.planet-noe.org.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision based approach. *Journal of Artificial Intelligence Research* 4:287–339.

Rintanen, J. 1999. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research* 10:323–352.

Son, T., and Baral, C. 2001. Formalizing sensing actions. a transition function based approach. *Artificial Intelligence* 19–91.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty and sensing actions. In *Proceedings of AAAI-98*.

Wilkins, D. E. 2001. A call for knowledge-based planning. *Artificial Intelligence Magazine* 22.