The Nature of Knowledge in an Abductive Event Calculus Planner

Leliane Nunes de Barros
leliane@ime.usp.br
Depto. de Ciência da Computação
Instituto de Estatística e Matemática
Universidade de São Paulo
Rua do Matão, 110
05508-900 - São Paulo - Brazil
Phone: (55)(11) 818 6235 FAX: (55)(11) 818 6134

Paulo E. Santos
santos@lsi.usp.br
Depto. de Engenharia Elétrica
Escola Politécnica
Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, n.158
05508-900 - São Paulo - Brazil
Phone: (55)(11) 818 5530

Abstract

There are several works whose goal is to specify complete and sound planning systems based on general purpose theorem provers. Some planners implemented in this way can have a close correspondence with existing partial-ordered planning algorithms. To improve the efficiency of logic-based planners we would like to use some of the results achieved by the AI planning community over the past twenty years in terms of algorithm design. We claim that a *knowledge level* analysis of *problem-solving methods* for planning, can help to identify what is the role of each piece of knowledge in a system and provide a common language to map, classify and compare different systems. In this paper we analyze an abductive event calculus planner using a library of problem-solving methods for planning.

1 Introduction

The main motivation for the construction of logic-based planners is the possibility to specify planning systems in terms of general theories of action and implement them as general purpose theorem provers without worrying about algorithmic concerns. A planning system defined in this way has a close correspondence between implementation and specification. Although this is a very important feature in the design of a planner the main issue of the planning task still remains: how to reduce the search space size?

There are several works aiming the construction of sound and complete logic-based planning systems [7] [17] [12] [8]. Moreover, some recent research results [6] demonstrate that, a good theoretical solution can co-exist with a good practical solution despite of contrary widespread belief [13].

In this paper we are going to analyze a particular logic-based planner, named Abductive *Event Calculus Planner* (AEC) [6], which has *Circumscriptive Event Calculus* [16] as the *reasoning about action and change*, with abduction as the inference rule. Within this framework, the work in [6] shows that there is a close correspondence between well known planning algorithms and planning defined as a result of the application of general purpose theorem proving techniques to a general purpose action formalism. In addition of that it was also shown that, by adopting theorem proving techniques for planning, the efficiency results of AEC planner was not exponential but comparable to most of the practical planners.

However, we must not neglect results from the past twenty years concerning algorithmic design and efficiency. As a matter of fact, there are important lessons we can take from the analysis of hand-coded planners in order to improve the logic-based planners. To find a suitable correspondence between these two planning approaches, we have to bridge the gap between the algorithmic description and the theorem proving technique; in this paper we are going to cross this gap by assuming a common language capable of mapping, classifying and comparing different systems.

We claim that by using a knowledge level analysis [1] [25] for the planning task we may provide an interesting perspective for analyzing and comparing planning systems. One way to achieve it is by studying the way systems represent and use available knowledge to solve problems; in other words, which *problem-solving* (PSM) methods they use.

In this article we are going to use the results of our previous work related to the construction of a planning library of problem-solving methods available in the AI planning literature [1] [25] (Section 2) in order to guide the process of investigating the correspondence between different planning approaches. We analyze some aspects of Shanahan's Abductive Event Calculus (Section 3) in terms of how the knowledge is represented and used (Section 4) with the purpose of searching for possible improvements on its efficiency (Section 5).

2. A library of planning problem-solving methods

Modern approaches to Knowledge Acquisition (KA) stress the importance of having reusable modeling components to support a knowledge engineer in constructing a required system model. These components of knowledge can be organized into a library of problem-solving methods for a *generic task*, e.g. the planning task.

In addition to its standard role as a tool to help in engineering a knowledge-based system, a library of problem-solving methods for planning has another important role: it can be seen as a framework for representing and analyzing planning methods. The library defines a number of components (*knowledge roles* and basic *problem-solving methods*) to describe planning methods in a common language which makes it possible to map, classify and compare different methods.

The methods in the library [25] were obtained by describing planning methods available in the AI planning literature which characterizes our library as *system-derived* as opposed to *domain-derived* libraries [27]. These components adopt a division in two layers `CommonKADS` [15]. In the domain layer, a set of typical *knowledge roles* characterizes the main types of domain knowledge used in planning methods. They also help in understanding the way knowledge is structured by providing an index to the *domain models* used to play these roles. In the task layer, a *task-method decomposition structure* indexes a set of basic methods by defining in what ways a planning task can be decomposed (what we call *task structures*), and what decomposition results from applying the basic methods to those tasks. Detailed control knowledge is later added to these decompositions by means of *control structures*. The *task-method decomposition structure* proposes a characterization of planning methods in general, as distinct realizations of a basic decomposition of the planning task into three tasks: propose, critique and modify [4]. In order to connect the domain with the task layer, a set of *assumptions* (or *features*) is used to specify when a method is applicable to a domain or a problem, and vice-versa.

A number of versions of (parts of) a library of problem-solving methods for planning were discussed in: [21], where it is discussed in detail the knowledge roles and domain models; in [2], the basic methods and the task-method decomposition structure are presented in detail; in [3], we have focused on the knowledge acquisition benefits of the library, by concentrating on the *assumptions*; in [1] and [25], the problem-solving methods were extended to include control knowledge.

In the next subsections we present a brief description of the components that matter for the discussion in this paper: the knowledge roles and the task-methods for planning. For a more detailed description of the library of problem-solving methods for planning see [25].

A domain ontology for planning

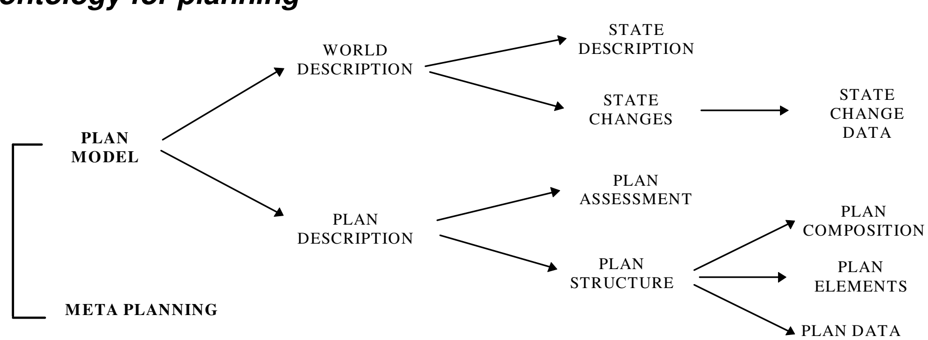

Figure 1 Hierarchy of static roles in planning

One of the critical elements in the analysis of a planning method is specifying the different *roles* that domain knowledge plays during the planning process. In the planning literature, domain knowledge is defined as static knowledge about the world which is only consulted during planning, but not manipulated. However, from a KA perspective, we must consider how this same knowledge is used by the planner itself in defining its dynamically changing model of the world. Thus, we identify two roles for domain knowledge: static and dynamic [15].

Static knowledge roles in planning

Figure 1 shows the hierarchical organization of static knowledge roles for planning [1]. The leaves of the hierarchy of static roles are associated with the types of domain knowledge (domain models) that can play these roles. The plan model role defines what a plan is and what it is made of. It consists of two parts: a world description and a plan description.

The **world description** role describes the world in which planning occurs. It is comprised of two sub-roles. (1) The state description role, which contains the knowledge necessary to represent or describe the state of the world (for example, a set of first order predicates as in STRIPS or a set of fluents from the Situation Calculus [10] or from the Event Calculus [26]). (2) The state changes role explicates the information connected to the specification of changes in the state of the world, i.e., events or actions (e.g., STRIPS-like operators; hierarchical task networks (HTNs) or the wwfs as the event calculus Initiates, Terminates and Releases formulae, that constitute a purely logical description of the effects of actions in a particular domain).

The **plan description** role describes the structure and features of the plan being generated, and comprises two sub-roles: plan structure and the (optional) plan assessment knowledge.

1. The **plan structure** role specifies how the parts of a plan (actions, sub-plans) are assembled together. It has two sub-roles: (a) the plan composition role, which describes whether the plan composition is total or partial order, whether it includes iteration and/or conditional operators, and whether the composition is hierarchical (i.e. whether plans can be recursively decomposed into *sub-plans*); (b) plan elements role describes the components of a plan that can be a state change, a goal or time point; (c) the plan data role contains additional information about the plan, ranging from bookkeeping information, to information about resource use in parts of the plan or yet interval constraints for binding the variables involved in the state changes. Two particularly important resources are agents and time.
2. The **plan assessment knowledge** role determines whether a certain plan (or sub-plan) is valid or whether a plan is better than another, in other words, it defines a theory of actions. A plan is valid if, after executed, it satisfies a list of desirable goals. Based on this knowledge, a plan can be constructed, criticized or modified. An example of plan assessment knowledge is a *rule-based-criterion* which can be used to find out if a condition is true at some (time) point in the plan (as in the *Modal Truth Criterion* (***mtc***) n TWEAK [5] or in the Event Calculus axioms [26]) (in the next sections we will call the *rule-based-criterion* as ***mtc-based***). Another example is the ***causal-link-based*** knowledge, also called *Protection Intervals*, which corresponds to a bookkeeping device for keeping track of the goals that have already been worked on, and thereby avoiding undoing them or working on them multiple times (as in SNLP [9]) The *causal-link based* model contains an explicit causal structure of the plan plus some evaluation criterions [23]. Both types of knowledge representation for the plan assessment role can be used to answer questions about the plan as: is a desired goal *g* satisfied by the plan? if it is not, what is wrong with the plan? how can we change the plan in order to fix it?

Dynamic roles in planning

Dynamic knowledge roles characterize planning in terms of the relevant variables whose values are constantly updated during the planning process. The dynamic roles include: (1) The ***current state*** role, which is initially filled by a description of the world at the beginning of the plan, but is subsequently modified to represent intermediary states in the plan. (2) The ***goal*** role, which describes the active goal or subgoal being worked on by the planner. The content of ***goal*** can be a set of conditions or a set of actions to be accomplished. Initially, this role is linked to the original goal, and during planning it may be updated with subgoals or decompositions of the original goal. (3) The ***conflict*** role contains the result of checking the plan for inconsistencies with respect to its conditions. (4) The ***plan*** role is a composite role consisting of (a) ***plan-steps***, (b) ***ordering constraints*** over the plan-steps, (c) ***variable binding constraints***, and (d) ***auxiliary constraints*** which correspond to a supporting knowledge used by a planner to reason about the construction of a plan and that it is not part of the plan solution. That is the case of ***causal-links*** or ***protected intervals*** which represent temporal and truth constraints between plan-steps and conditions.

The tasks and methods for planning

Based on an analysis of many classical planning systems, we have identified relevant tasks and problem-solving methods. We organize these into a task-method decomposition structure [11] in Figure 2 (where ellipses represent tasks and rectangles methods). A method *executes* (solid lines) a number of subtasks and a (sub)task can be *performed by* alternative (dashed lines) methods. The leaves of the task-method tree are called *primitive-methods* and the tasks they perform, *primitive-tasks*. Methods have two additional types of knowledge associated with them: control knowledge and suitability criteria [1].

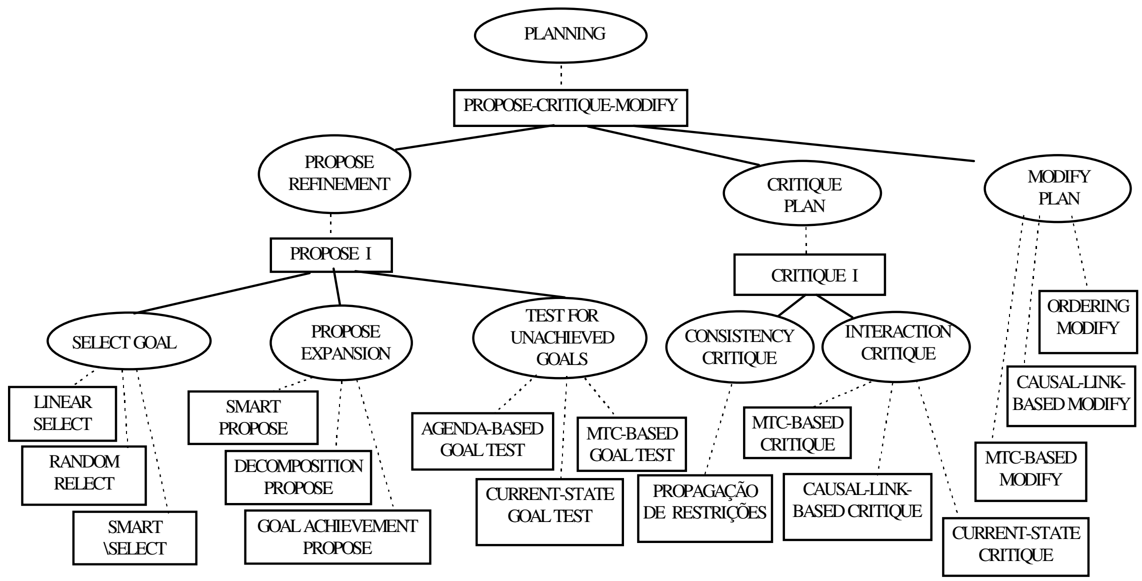

Figure 2 The task-method decomposition structure for the planning task. Solid lines stand for *executes* (a method executes its subtasks), dashed lines denote *performed-by* (a task can be performed by alternative methods).

The class of planners we are dealing with share a general, high-level problem-solving method called propose-critique-modify (PCM) [4]. That is, the planners all contain in one way or another these three basic tasks: (i) propose expansion, (ii) critique plan and (iii) modify plan. Planners differ in the problem-solving methods (PSMs) they use to perform these three tasks. These differences also reflect how planning knowledge is represented. For example, in Figure 2, the propose-I method consists of the three subtasks: select goal, propose expansion, and test for unachieved goals. The propose expansion task can, in its turn, be realized by three different methods: smart propose, goal achievement propose, and decomposition propose, and so on. For a detailed description of all tasks and methods involved see [25]. In Section 4 we describe in more details the subtasks/methods realized by the Abductive Event Calculus Planner.

The above framework gives a better understanding of what are the possible choices for a planner design in terms of the nature of the knowledge they use, how they are represented and their choices on control structures.

3. Overview on Abductive Event Calculus Planner

Roughly speaking, abduction is the process of explaining a sentence Γ by finding a set of formulae Δ (called *residue*) such that, given a set of formulae Σ (background theory), Γ is a logical consequence of the conjunction of Σ and Δ [6]. This process of *finding explanations* to goals is applied to a reasoning about actions and state change formalism: *Event Calculus* (EC). Intuitively, Event Calculus, is a temporal (first-order) formalism designed to model and reason about scenarios described as a set of *events* whose occurrences have the effect of starting or terminating the validity of determined properties of the world (the *fluents*). There are many versions of Event Calculi [19]. In this paper we are going to work with the variant of the calculus defined by Shanahan in [16].

The Event Calculus with Circumscription

Below we briefly present the axiomatization of the Circumscriptive Event Calculus [16] and, in the sequence, we define planning as a process of abduction on this event (action) theory.

The Event Calculus is a 3-sorted first-order language comprising a sort of time points (t_1, t_2, ..., t_n, t_{n+1}, ...); a sort of fluents (f_1, f_2, ..., f_n, f_{n+1}, ...); and, a sort of event types (a_1, a_2, ..., a_n, a_{n+1}, ...). The following are the predicates of the Event Calculus:

HoldsAt(f,t)	Fluent *f* holds at time *t*;
Initially(f)	Fluent *f* holds from time *0*;
Happens(a,t)	Event *a* happens at time *t*;
Initiates(a,f,t)	Fluent *f* holds after the occurrence of an event *a* at time *t*;
Terminates(a,f,t)	Fluent *f* does not hold after the occurrence of an event *a* at time *t*;
Releases(a,f,t)	Fluent *f* is not subject to the common-sense law of inertia after event *a* at time *t*;
Clipped(t_1,*f*,t_2)	Fluent *f* ceased to hold (or it is released from the common sense law of inertia) in some time point in [t_1, t_2];
Declipped(t_1,*f*,t_2)	Fluent *f* started to hold (or it is released from the common sense law of inertia) in some time point in [t_1, t_2].

The predicates described above are constrained by the following axioms:

$$HoldsAt(f,t) \leftarrow Initially(f) \wedge \neg\, Clipped(0,f,t) \quad \text{(EC1)}$$

$$HoldsAt(f,t_2) \leftarrow Happens(a,t_1) \wedge Initiates(a,f,t_1) \wedge t_1 < t_2 \wedge \neg Clipped(t_1,f,t_2) \quad \text{(EC2)}$$

$$\neg HoldsAt(f,t_2) \leftarrow Happens(a,t_1) \wedge Terminates(a,f,t_1) \wedge t_1 < t_2 \wedge \neg Declipped(t_1,f,t_2) \quad \text{(EC3)}$$

We have to consider also the following defined notions:

$$Clipped(t_1,f,t_2) \leftrightarrow \exists\, a,t\, [Happens(a,t) \wedge [Terminates(a,f,t) \vee Releases(a,f,t)] \wedge t_1 < t \wedge t < t_2] \quad \text{(EC4)}$$

$$Declipped(t_1,f,t_2) \leftrightarrow \exists\, a,t\, [Happens(a,t) \wedge [Initiates(a,f,t) \vee Releases(a,f,t)] \wedge t_1 < t \wedge t < t_2] \quad \text{(EC5)}$$

The frame problem is overcome through circumscription. Given a conjunction Σ of formulae, not including the predicates *Initially* or *Happens*, describing the effects of actions (*domain description*), a conjunction Δ of formulae, not including the predicates *Initiates*, *Terminates* or *Releases*, describing the *narrative* of actions and events, a conjunction Ω of *uniqueness-of-names* axioms for actions and fluents, a conjunction EC of the axioms of the Event Calculus ((EC1) to (EC5)), we have to consider the following formula:

$$\mathrm{CIRC}[\Sigma; Initiates, Terminates, Releases] \wedge \mathrm{CIRC}[\Delta; Happens] \wedge \mathrm{EC} \wedge \Omega$$

where CIRC[Σ; α, β] means the circumscription of Σ with relation to the predicate symbols α and β. By circumscribing *Initiates*, *Terminates* and *Releases* we are imposing that the known effects of actions are the only effects of actions, and by circumscribing *Happens* we assume that there are no unexpected event occurrences. A more thorough discussion of it can be found in [18].

Planning in Event Calculus

Planning in Event Calculus is naturally handled as an abduction process [7]. In this setting, planning a sequence of actions in order to satisfy a given goal Γ, given a domain description Σ is the abduction explanation Δ such that $\Sigma, \Delta \models \Gamma$; which becomes with the circumscription policy stated above:

$$\mathrm{CIRC}[\Sigma; Initiates, Terminates, Releases] \wedge \mathrm{CIRC}[\Delta; Happens] \wedge \mathrm{EC} \wedge \Omega \models \Gamma$$

In other words, Δ is a plan for Γ . In [6] a planning system based in these ideas is presented as a PROLOG meta-interpreter, where each of the EC axioms were encoded in meta-level. In the next sections we will analyze the EC planner based on the planning library. A complete discussion of the meta-interpreter can be found in [6].

4. The nature of knowledge in the AEC planner

The Abductive Event Calculus Planner is an abductive theorem prover, coded as a PROLOG meta-interpreter. The job of an abductive meta-interpreter is to construct a *residue* of *abducible* literals that can not be proved from the object-level program [6] which, in the case of a planning task, corresponds to the *plan description*.

Static knowledge roles in the AEC planner

Knowledge role	**Domain model**
state description	a set of fluents (f_1, f_2, ..., f_n, f_{n+1}, ...)
state changes	action α is described through a conjunction of event calculus formulae of the form: $Initiates(\alpha,f,t) \leftarrow \Pi$ or $Terminates(\alpha,f,t) \leftarrow \Pi$ or $Releases(\alpha,f,t) \leftarrow \Pi$, where Π is of the form, $(\neg)\ HoldsAt(f_1, t) \wedge .... \wedge (\neg)\ HoldsAt(f_n, t)$
plan composition	partial order plan
plan elements	*happens* (α, t) literals and time points ordered by *before*(t_1,t_2) literals
plan data (causal-links or protected intervals)	negated *clipped* or *declipped* literals
plan assessment knowledge	1. *mtc-based*: the event calculus axioms encoded in the PROLOG meta-interpreter. 2. *causal-link-based*: as the process to preserve the list of negated *clipped* or *declipped* literals (causal-links) as part of the abductive process.

Table 1 Static knowledge roles in the AEC planner.

Table 1 shows how the planning knowledge roles are fulfilled in the AEC planner. The *state description* role is played by a set of fluent terms of the domain. The *state changes* (actions or events) are played by a conjunction of event calculus formulae involving $Initiates(\alpha,f,t)$, or $Terminates(\alpha,f,t)$ or $Releases(\alpha,f,t)$ of the object level program. Although time points are used as an additional *state change data*, the AEC is not a temporal planner: the time points are used in a similar way as the temporal ordering in classical planning where ordering constraints are imposed over plan-steps. The time points used to describe *state changes* have a similar meaning as in a STRIPS-like action: it is used to describe what are the fluents that must hold ***after*** (effects) or ***before*** (preconditions) the action's execution.

The *plan elements* correspond to the abducible literals *happens* and *before*. Besides of that, time points also play the role of *plan elements*: they will be used to describe the execution ordering of the actions in the plan solution (like the *plan-steps* used in most of non-temporal planning systems [1]).

The *plan composition* is a partial order over the *plan elements* since an order between time points is only imposed when an action is added to be *before* a certain goal or in order to fix a threat in the plan. The *plan data* is played by the literals *clipped* and *declipped* as additional information (bookkeeping information) to be used by the planning tasks.

The *plan assessment knowledge* role is represented in the AEC planner through two different domain models: as a *mtc-based* model and as a *causal-link-based* model. As pointed out by [8], the EC axioms have a close correspondence with Chapman's modal truth criterion [5] and since the theorem prover is a

[1] In the EC presented in [6] only one action can occur in the same time point *t*, therefore there is a correspondence between time point and plan-step from classical planning.

compilation of the event calculus axioms, the *mtc-based* model of the *plan assessment knowledge* role is encoded as meta-clauses of the PROLOG meta-interpreter. On the other hand, the *causal-link-based* model is implemented as the process of preserving the list of negated *clipped* or *declipped* literals (causal-links) as part of the abductive process (as we will se bellow).

Most of the systems from the AI-planning literature represents *plan assessment knowledge* in one way or another way (with the exception of NONLIN [24]). However, there is no harm on having different representations for the same knowledge in a planner.

Task-method decomposition in the AEC planner

Using the planning library we will characterize the problem-solving method employed by the AEC planner. Although the specification of a theory of actions (*plan assessment knowledge*) is preserved in the implementation (each meta-level clause correspond to one EC axiom), it is not immediately clear how subtasks and methods are performed unless we make a close analysis of the PROLOG program. Figure 3 shows the *abdemo* clause (standing for abductive demonstration) which is the description of the EC2 axiom in the meta-level.

```
<Line 1> abdemo ([holds_at (F1,T3)|Gs1],R1,R5,N1,N4) :-
<Line 2>          abresolve (initiates (A,F1,T1),R1,Gs2,R1),
<Line 3>          abresolve (happens (A,T1,T2),R1,[],R2),
<Line 4>          abresolve (before (T2,T3),R2,[],R3),
<Line 5>          append (Gs2,Gs1,Gs3),
<Line 6>          add_neg ([clipped (T1,F1,T3)],N1,N2),
<Line 7>          abdemo_nafs (N2,R3,R4,N2,N3),
<Line 8>          abdemo (Gs3,R4,R5,N3,N4).
```

Figure 3 The EC2 axiom encoded in meta-level PROLOG program. In order to demonstrate (abductively) that a fluent F1 holds_at time T3, the meta-interpreter has to demonstrate the 7 preconditions of the clause.

Planning subtask	Problem-solving method
select goal	*linear select*
test for unachieved goals	*Agenda-based goal test*
propose expansion	*goal achievement propose*
interaction critique	*Causal-link-based critique*
modify plan	*Causal-link-based modify*

Table 2 Subtasks and corresponding methods performed by the AEC planner.

Table 2 shows the five subtasks being executed by the AEC planner and the corresponding methods used to perform them as we will describe bellow:

Select goal This task selects a goal from the set of goals to be accomplished which is performed by the *linear select* method in the AEC planner: it selects the last goal established by its search strategy (goals are added on the beginning of a PROLOG list and are selected as a LIFO goal ordering strategy).

Test for unachieved goals This task checks the current plan for unachieved goals. The method used by the AEC planner to realize this task is the *agenda-based goal test*, i.e., the planner checks if the list of goals is not empty, otherwise the planning task is finished. Only planners that exploit *causal-links* can use this method once they guarantee that the goals already processed are protected by a causal-link. This is because processed goals, that are extracted from the list of goals, can possibly be clobbered during the planning process if the planner do not protect the goals already achieved.

Propose expansion The AEC planner performs this task by a *goal-achievement propose* method, like in most of the partial-order planners. In Figure 3, the PROLOG meta-interpreter clause corresponds to the main part of this method. This task takes the selected goal (Line 1), and proposes a way to accomplish it by selecting an action whose effect includes the goal (Line 2 and Line 3 inserts *happens*(A, T1, T2) in the residue R1 obtaining the residue R2), constraining the place of the action in the plan to be *before* the time point (T3) when the selected goal needs to be valid (Line 4). When a new action is added to the plan, its preconditions are added to the set of goals (Line 5). When there is already a step in the plan that achieves the goal, only the ordering constraint is added to the plan (this is done by a special clause *abresolve* for the *before* predicate (Line 4)). Within this method, planners that protect its goals add a causal-link for the just accomplished goal (Line 6 adds a *clipped*(T1, F1, T3) predicate in a list of facts that can never be satisfied in the proof process). It is interesting to notice that this method has not been characterized in the planning library as *mtc-based* or *causal-link-based* .

Interaction critique This task verifies whether the proposed action for accomplishing the goal would interact with other goals in the plan (e.g., one action might undo (clobber) the precondition of another action). For realizing the interaction critique task, the AEC planner uses the *causal-link-based critique* method, which checks if the proposed plan-step, threats any existent causal-link. Notice that the original EC2 axiom do not define that it is necessary to *protect the interval* defined in between the starting time of the action and the instant in which the fluent hold but only to demonstrate that the interval is not being clobbered by another action, that is, if the negation of *clipped*(T1, F1, T3) holds. The interval protection is done as a part of the abductive process by the meta-predicates *add_neg*([*clipped*(T1,F1,T3)],N1,N2) and *abdemo_nafs*(N2,R3,R4,N2,N3) where the former adds the formula *clipped*(T1,F1,T3) to the list of negated goals, if it is not already there, while the later is only satisfied if there is not any possibility to resolve *clipped*(T1,F1,T3).

Modify plan This task is responsible for modifying the plan with respect to the results of the critique plan subtask (a conflict). A plan modification can be done by adding ordering or variable bindings to the plan until a conflict is solved or avoided. The AEC planner uses the *causal-link-based modify* method, which adds to the plan an ordering constraint (a *before* literal) before or after the protected causal-link (*clipped* literal).

The interaction critique subtask is performed each time the predicate *abdemo_nafs*(N2,R3,R4,N2,N3) is called (Line 7 in Figure 3). This is done by verifying whether any of the elements of the list of negated goals N2 actually fails given R3 (the plan). If not, then there is a threat that should be solved by *promotion* or *demotion* which are additions to the residue to preserve the proof of a previously solved negated clipped goal [6]. This is done by a special clause that defines the *abdemo_nafs* predicate (see [6]).

5. A lesson taken from the analyses of partial-order planning algorithms

In this section we show that, since we have established a common language to describe planners, we can look at comparative analyses of planning algorithms and try to map its results for the logic based approach, e.g., for the AEC planning. In order to make this map, we use as an example an interesting experimental result pointed out by [22] which investigates the performance of partial-order planning algorithms based on the way they balance the tradeoff between redundancy and systematicity of its search.

The tradeoff between redundancy and systematicity

There was a belief that by decreasing redundancy it would be possible to improve planning efficiency. Designing a planner that is systematic means that it never visits two equivalent plans, i.e., it does not allow redundancy in its search space [9]. However, Kambhampati in [22] has shown that there is a tradeoff between redundancy elimination and least commitment: redundancy is eliminated at the expense of increasing commitment in the planner. Therefore, the performance of a partial-order planner is better predicted based on the way it deals with the tradeoff between redundancy and commitment, then on the systematicity of its search as claimed in [9].

In order to show the effects of this tradeoff, Kambhampati starts by choosing two well known planning algorithms named TWEAK [5] and SNLP [9]. They can be characterized as *mtc-based* and *causal-link-based* planners, respectively.

Redundancy in a MTC-based planner TWEAK is a planner achieved by inverting the modal truth criterion [5]. TWEAK does not keep track of which goals were already achieved and which it remains to be achieve, it may achieve and clobber a goal/subgoal arbitrarily many times. Thus, TWEAK turns out to have a lot of redundancy in its search space.

Systematicity in a causal-link based planner SNLP [9] achieves systematicity by keeping track of the causal structures of the plans generated during search, and ensuring that each branch of the search space commits to and protects mutually exclusive causal structures for the partial plans [22]. Such protection corresponds to a strong form of premature commitment, which can increase the amount of backtracking as well as the solution depth, and can have an adverse effect on the performance of the planner.

Kambhampati's experimental analyses shows that there is a spectrum of solutions to the tradeoff between redundancy and commitment in partial-order planning (Figure 4 in [22]) in which the SNLP and TWEAK planners fall in the extremes. He also proposes other planners that can fall in the middle of these two extremes and can have a better performance, for instance, a *multi-contributor causal-link* planner. In the sequence we will discuss how such planner can be implemented using the abductive event calculus approach.

How the AEC planner is related to TWEAK and SNLP

Kambhampati classifies PO planners as *Causal Link Based* or *Non-Causal Link Based*. Since the AEC planner does bookkeeping of the *clipped* goals, it can be characterized as a causal-link based planner as in SNLP planner (extreme systematic).

On the other hand, the AEC planner uses the *plan assessment knowledge* role modeled as a set of Event Calculus axioms which corresponds to the MTC knowledge in TWEAK (extreme redundant).

The reason why the AEC planner should still be considered a systematic planner (like SNLP) is because in the AEC planner's design a decision was made to add as an *auxiliary constraint* the literal *clipped*(T1, F, T2) in the list of *negative residue*. If instead, the AEC planner added the literals *holds_at*(T1, F, T2) it would become extremely redundant (like in TWEAK).

An abductive event calculus planner with multi-contributor causal-links

The AEC planner, such as SNLP and NONLIN, uses the traditional single contributor causal-links and fix a threat (conflict) by ordering it to come either before or after the causal-link (demotion or promotion). If none of those conflict resolution methods can be applied, the planner backtracks. On the other hand, a planner bookkeeping *a list of all possible contributors within the causal-links* can reduce premature commitment with a single-contributor causal-link. As showed in [22], this sacrifices systematicity but reduces backtracking.

In the AI planning terminology, a *multi-contributor causal-link* is a 3-tuple <*S, p, w*> where *w* is a plan-step which requires the condition *p* to be true; *S* is a **set of steps** belonging to the plan that satisfy the condition *p* .

We can change the AEC planner into a *multi-contributor causal-link* planner. In order to do so, the abductive meta-interpreter has to look for ***all the actions*** in *happens* that satisfy a fluent *F* and keep a causal-link (*clipped* literal) for each of those actions (for instance, *clipped*(T1, F, T) for a *happens* (A1, T1, Ti) , *clipped*(T2, F, T) for a *happens* (A2, T2, Tj), ... , *clipped*(Tn, F, T) for a *happens* (An, Tn, Tm)).

By keeping this list we will have to make changes in the AEC planner with respect to the way the planner realizes its planning subtasks, i.e., the PSMs. The specification of those methods is out of the scope of the present article.

6. Concluding Remarks

The AI planning community has developed very different planning systems using apparently unrelated data structures and algorithms. Much research has been done to analyze and compare them in terms of efficiency gains and yet in terms of the nature of the knowledge they use.

In previous work we have constructed a library of problem-solving methods for planning [25]. The methods in the library were obtained by describing planning methods available in the AI planning literature. A central knowledge in the planning library is the knowledge role called *plan assessment knowledge* which has been identified to be modeled as: *causal-link-based* or *mtc-based*. The methods used to solve planning subtasks can also be characterized as *causal-link-based* or *mtc-based*, that means, a problem-solving method can use the *plan assessment knowledge* represented as *causal-link-based* or *mtc-based* respectively.

We have analyzed Shanahan´s AEC planner [6] in terms of the knowledge it represents and how it performs the subtasks described in the planning library. In the AEC planner, the *plan assessment knowledge* role is represented in both ways: as a *mtc-based* model (a compilation of EC axioms into the clauses of the abductive meta-interpreter) and as a *causal-link-based* model (in the abductive process of preserving the list of negated *clipped* or *declipped* literals (causal-links)).

The interesting feature pointed out by Shanahan [6] is that the planning knowledge components (as conflict, causal-links, promotions and demotions) were not originally designed in the logic programming implementation but rather they were features that naturally arise of the theorem prover's search for a proof. Such affirmative can be considered partially true: the idea of having a list of *clipped* goals to be recorded by a bookkeeping procedure do not follow directly from the EC axioms execution but is rather a design choice of the theorem prover implementation.

Therefore, the choices on the contents of the residue list can determine whether the planner is *causal-link-based* or *mtc-based* (which in terms of search space characterization means redundant or systematic [22]). If instead of bookkeeping the *clipped* literal in the list of negated residue, the AEC planner had, on the other hand, bookkeeping the *holds-at* literal, the planner would be redundant (like TWEAK [5]). This is still a nice feature of the logic based planner since it shows a clean and simple distinction between the so called *causal-link-based* and *mtc-based* planning methods.

Based on comparative analyses of planning systems [22] we have also proposed a way in which the AEC planner can be changed to become more efficient: bookkeeping *multi-contributors causal-links* in the list of residue. This can be considered a more controlled way to improve efficiency in a logic-based planner instead of inserting a number of *cuts* in the PROLOG program. It also may highlight new ideas on the extension of a theory of action, e.g., the Event Calculus.

References

[1] L. N. Barros, J. Hendler, and V. R. Benjamins. Par-KAP: a knowledge acquisition tool for building practical planning systems. In *Proceedings of the IJCAI-97*, Nagoya, Japan, 1997.

[2] L. N. Barros, A. Valente and V. R. Benjamins. Modeling planning tasks. In *Third International Conference on Artificial Intelligence Planning Systems, AIPS-96,* pages 11-18, 1996.

[3] V. R. Benjamins, L. N. Barros, and A. Valente. Constructing planners through problem-solving methods. In *Knowledge Acquisition Workshop - KAW'96 (Banff)*, 1996.

[4] B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine,* 11: (59-71), 1990.

[5] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333-377, 1987.

[6] M. Shanahan. An Abductive Event Calculus Planner. To appear. http://www-ics.ee.ic.ac.uk/~mpsha/pubs.html.

[7] K. Eshghi. Abductive planning with event calculus. In *Proceedings of the Fifth International Conference on Logic Programming,* 562-579, 1988.

[8] L. Missiaen, M. Bruyonooghe, and M. Denecker. Chica, a planning system based on event calculus. *The Journal of Logic and Computation,* 5(5): 579-602, 1995.

[9] D. McAllester and D.Rosenblitt. Systematic Nonlinear Planning. In Proceedings of 9th AAAI, 19991.

[10] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In *Machine Intelligence 4*, 1969.

[11] K. Orsvärn. Principles for libraries of task decomposition methods - conclusions from a case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Lectures Notes in Artificial Intelligence, 1076, 9th European Knowledge Acquisition Workshop, EKAW-96,* pages 48-65. Springer-Verlag, 1996.

[12] R. Reiter. Knowledge in action: Logical foundations for describing and implementing dynamical systems. Draft of the first eight chapters of the book.

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall International, 1995.

[14] P. E. Santos. Formalizing the common sense of a mobile robot. Mol-1998-02, Institute for Logic, Language and Computation, University of Amsterdam, September 1998.

[15] A. Th. Schreiber, B. J. Wielinga, R. de Hoog, J. M Akkermans, and W. Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert,* 9(6): 28-37, December 1994.

[16] M. Shanahan. A Circumscriptive event calculus. *Artificial Intelligence*, 77:249-284, 1995.

[17] M. Shanahan. Event Calculus planning revised. In *Proceedings of European Conference on Planning,* pages 390-420. Springer-Verlag Lecture notes in Artificial Intelligence 1348, 1997.

[18] M. Shanahan. Solving the frame problem. The MIT press, 1997.

[19] M. Shanahan. A logical account of the common sense information situation for a mobile robot. Linköping Electronic Articles in Computer and Information Science, 1998. http://www.ep.liu.se/ea/cis/1998/.

[20] A. Tate, J.Hendler, and M.Drummond. A review of AI planning techniques. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning,* pages 26-49. Kaufmann, San Mateo, CA, 1990.

[21] A. Valente. Knowledge level analysis of planning. SIGART Bulletin, 6(1): 33-41, 1995.

[22] S. Kambhampati. On the Utility of Systematicity: Understanding Tradeoffs between Redundancy and Commitment in Partial-order Planning. In *Proceedings of IJCAI*, 1991.

[23] S. Kambhampati. Multi-Contributor Causal Structures for Planning: A Formalization and Evaluation. *Artificial Intelligence*, Vol. 69, 1994.

[24] A. Tate. Generating Project Networks. In *Proceedings of IJCAI-77*, pages 888-893, Boston, 1977.

[25] A. Valente, V. R. Benjamins, and L. N. Barros. A library of system-derived problem-solving methods for planning. In *International Journal of Human-Computer Studies 48,* 417-447. Academic Press Limited, 1998.

[26] R. A. Kowalski and M. J. Sergot. A Logic-Based Calculus of Events. In *New Generation Computing.* Vol. 4:67-95, 1986.

[27] H. Cottam and N. Shadbolt.Domain and system influences in problem solving models for planning. In N. Shadbolt, K. O'Hara, and G. Schreiber, editors, *Lectures Notes in Artificial Intelligence, 1076, 9th European Knowledge Acquisition Workshop, EKAW-96,* pages 354-369. Springer-Verlag, 1996.

[28] R. E. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence,* 2:(169-203), 1971.

[29] A. Barret and D. S. Weld. Partial-order planning: evaluating possible efficiency gains. *Artificial Intelligence,* 67: (71-112), 1994.