Planning with Incomplete Information

Antonis Kakas
University of Cyprus
antonis@cs.ucy.ac.cy

Rob Miller
University College, U.K.
rsm@ucl.ac.uk

Francesca Toni
Imperial College, U.K.
ft@doc.ic.ac.uk

Abstract

Planning is a natural domain of application for frameworks of reasoning about actions and change. In this paper we study how one such framework, the Language $\mathcal{E}$, can form the basis for planning under (possibly) incomplete information. We define two types of plans: *weak* and *safe* plans, and propose a planner, called the *$\mathcal{E}$-Planner*, which is often able to extend an initial weak plan into a safe plan even though the (explicit) information available is incomplete, e.g. for cases where the initial state is not completely known. The $\mathcal{E}$-Planner is based upon a reformulation of the Language $\mathcal{E}$ in argumentation terms and a natural proof theory resulting from the reformulation. It uses an extension of this proof theory by means of abduction for the generation of plans and adopts argumentation-based techniques for extending weak plans into safe plans. We provide representative examples illustrating the behaviour of the $\mathcal{E}$-Planner, in particular for cases where the status of fluents is incompletely known.

Introduction

General formalisms of action and change can provide a natural framework for the problem of planning. They can offer a high level of expressivity and a basis for the development of general purpose planning algorithms.

We study how one such formalism, the Language $\mathcal{E}$ (Kakas & Miller 1997b; 1997a), can form a basis for planning. To do this we exploit the reformulation (Kakas, Miller, & Toni 1999) of the Language $\mathcal{E}$ into an argumentation framework and the associated proof theory offered by this reformulation. A simple extension of this argumentation-based proof theory with abduction forms the basis of planning algorithms within the framework of the Language $\mathcal{E}$.

In this paper we will be particularly interested in addressing the specific problem of planning under incomplete information. This amounts to planning in cases where some information is missing, as for example when we do not have full knowledge of the initial state of the problem. In general, we assume that this missing information cannot be "filled in" by additional actions in the plan as it may refer to properties that cannot be affected by any type of action in the theory or to an initial time before which no actions can be performed. Instead, the planner needs to be able to reason despite this incompleteness and construct plans where this luck of information does not matter for achieving the final goal.

We define a planner, call the $\mathcal{E}$-Planner, which is able to solve this type of planning problems with incomplete information. It works by first generating a conditional plan based on one possible set of arguments in the corresponding argumentation theory of the planning domain. These plans are called weak plans and may not be successful under every possibility for the missing information. The planner then uses further argumentation reasoning to extend the weak plan to a safe plan which is able to achieve the planning goal irrespective of the particular status of the missing imformation.

Planning under incomplete information is a relatively new topic. In (Finzi, Pirri, & Reiter 1999) this problem is called "Open World Planning" and is studied within the framework of the situation calculus. The incomplete information refers to the initial situation of the problem and a theorem prover is used to reason about properties at this situation. Other related work on planning within formal frameworks for reasoning about actions and change is (Levesque 1996), which defines a notion of conditional plans, (Shanahan 1997; Denecker, Missiaen, & Bruynooghe 1992), with a formulation of abductive planning in the event calculus and (Dimopoulos, Nebel, & Koehler 1997; Lifschitz 1999), which study "answer set planning" within extended logic programming.

A Review of the Basic Language $\mathcal{E}$

The Language $\mathcal{E}$ is really a collection of languages. The particular vocabulary of each language depends on the domain being represented, but always includes a set of *fluent constants*, a set of *action constants*, and a partially ordered set $\langle \Pi, \preceq \rangle$ of *time-points*. For this paper where we are interested in linear planning we will assume that $\preceq$ is a total order. A *fluent literal* is either a fluent constant F or its negation $\neg F$.

Domain descriptions in the Language $\mathcal{E}$ are collections of statements of three kinds (where A is an action constant, T is a time-point, F is a fluent constant, L is a fluent literal and C is a set of fluent literals): *t-propositions* ("t" for "time-point"), of the form L `holds-at` T; *h-propositions* ("h" for "happens"), of the form A `happens-at` T; *c-propositions* ("c" for "causes"), of the form A `initiates` F `when` C or A `terminates` F `when` C. When C is empty, the c-propositions are written as "A `initiates` F" and "A `terminates` F", resp.

The semantics of $\mathcal{E}$ is based on simple definitions of interpretations, defining the truth value of t-propositions at each particular time-point, and models. Briefly, (see (Kakas & Miller 1997a; 1997b) for more details) these are given as follows:

- An *interpretation* is a mapping $H : \Phi \times \Pi \mapsto \{true, false\}$, where Φ is the set of fluent constants and Π is the set of time-points in $\mathcal{E}$. Given a set of fluent literals C and a time-point T, an interpretation H *satisfies* C *at* T iff for each fluent constant $F \in C$, $H(F,T) = true$, and for each fluent constant F' such that $\neg F' \in C$, $H(F',T) = false$.
- Given a time-point T, a fluent constant F and an interpretation H, T is an *initiation-point* (*termination-point* resp.) *for* F *in* H *relative to* a domain description D iff there is an action constant A such that (i) D contains both an h-proposition A `happens-at` T and a c-proposition A `initiates` (`terminates`, resp.) F `when` C, and (ii) H satisfies C at T. Then, an interpretation H is a *model* of a given domain description D iff, for every fluent constant F and time-points $T_1 \prec T_3$:
  1. If there is no initiation- or termination-point T_2 for F in H relative to D such that $T_1 \preceq T_2 \prec T_3$, then $H(F,T_1) = H(F,T_3)$.
  2. If T_1 is an initiation-point for F in H relative to D, and there is no termination-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F,T_3) = true$.
  3. If T_1 is a termination-point for F in H relative to D, and there is no initiation-point T_2 for F in H relative to D such that $T_1 \prec T_2 \prec T_3$, then $H(F,T_3) = false$.
  4. For all t-propositions F `holds-at` T in D, $H(F,T) = true$, and for all t-propositions "$\neg F$ `holds-at` T'" in D, $H(F,T') = false$.
- A domain description D is *consistent* iff it has a model. Also, D *entails* (written $\models$) the t-proposition F `holds-at` T ($\neg F$ `holds-at` T, resp.), iff for every model H of D, $H(F,T) = true$ ($H(F,T) = false$, resp.).

Note that the t-propositions, in effect, are like "static" constraints that interpretations must satisfy in order to be deemed models. We can extend the language $\mathcal{E}$ with *ramification* statements, called *r-propositions*, of the form L `whenever` C, where L is a fluent literal and C is a set of fluent literals. These are also understood as constraints on the interpretations, but with the difference of being "universal", i.e. applying to every time point. Formally, the definition of a model is extended with:

5. For all r-propositions L `whenever` C, in D, and for all time-points T, if H satisfies C at T then H satisfies $\{L\}$ at T.

In addition, the complete formalization of ramification statements requires a suitable extension of the definitions of initiation- and termination-point. The interested reader is refered to (Kakas & Miller 1997a) for the details.

As an example, consider the following simple "car engine domain" D_c, with action constants *TurnOn* and *Empty* and fluents *Running* and *Petrol*:

TurnOn `initiates` *Running* `when` $\{Petrol\}$	(D_c1)
Empty `terminates` *Petrol*	(D_c2)
TurnOn `happens-at` 5	(D_c3)
Petrol `holds-at` 1	(D_c4)

It is easy to see, for example, that D_c entails *Running* `holds-at` 7 and that D_c extended via the h-proposition *Empty* `happens-at` 3 does not.

Planning with $\mathcal{E}$

The language $\mathcal{E}$ with its explicit reference to actions as h-propositions in its basic ontology is naturally suited for the problem of planning. Let a **goal** be a set of t-propositions. Then, given a domain description D and a goal G, planning amounts to constructing a set Δ of h-propositions such that $D \cup \Delta$ entails G.

In general, however, the extension of D via the plan Δ might be required to respect some given **preconditions** for the actions in Δ. These preconditions can be represented by a new kind of statements, called **p-propositions** ("p" for "preconditions"), of the form A `needs` C, where A is an action constant and C is a non-empty set of fluent literals. Intuitively, the fluents in C are conditions that must hold at any time that the action A is performed. Note that, alternatively, preconditions could be encoded via additional conditions in c-propositions already appearing in the domain descriptions. The use of p-propositions is though simpler and more modular.

Definition 1 *An* **($\mathcal{E}$-)planning domain** *is a pair $\langle D, P\rangle$, where D is a domain description and P is a set of p-propositions.*

The semantic interpretation of the new type of sentences is that of integrity constraints on the domain descriptions.

Definition 2 *Given a planning domain $\langle D, P\rangle$, D* **satisfies** *P, written $D \models P$, iff for all p-propositions A* `needs` *C in P, and for all h-propositions A* **happens-at** *T in D, D entails $C(T)$, where $C(T)$ denotes the set* *of t-propositions obtained by transforming every fluent literal in C into the respective t-proposition at T.*

The planning problem is then defined as follows.

Definition 3 *Given a planning domain $\langle D, P\rangle$ and a goal G, a* **(safe) plan for** G **in** D *is a set Δ of h-propositions such that $D \cup \Delta$ is consistent and :*
- $D \cup \Delta \models G$,
- $D \cup \Delta \models P$.

Note that the initial state of the planning problem is assumed to be contained in the given domain description, and might amount to a set of t-propositions at some initial time point, or, more generally a set of t-propositions over several time points, not necessarily all coinciding with a unique initial time point.

The above definition of (safe) plan provides the formal foundation of the $\mathcal{E}$-planner. It is easy to see that, through the properties of the model-theoretic semantics of $\mathcal{E}$, a safe plan satisfies the requirements that (i) it achieves the given goal, and (ii) it is executable.

As an example, let us consider the simple "car engine planning domain" $\langle D'_c, P_c\rangle$, with D'_c consisting of statements (D_c1), (D_c2) and (D_c4) from the previous section as well as:

$$\textit{Fill}\ \texttt{initiates}\ \textit{Petrol} \qquad (D_c5)$$
$$\neg \textit{Running}\ \texttt{holds-at}\ 1 \qquad (D_c6)$$

and P_c consisitng of the p-proposition

$$\textit{Fill}\ \texttt{needs}\ \neg\textit{Running} \qquad (P_c1)$$

Let the goal be $G = \textit{Running}\ \texttt{holds-at}\ T_f$ for some (final) time T_f. Then, a plan for G is given by the set $\Delta_1 = \{\textit{TurnOn}\ \texttt{happens-at}\ T_1\}$ where $T_1 \prec T_f$. This is a *safe* plan in the sense that if we add Δ_1 to D'_c, then both the goal G and the p-proposition in P_c are entailed by the augmented domain.

Consider now the domain D''_c obtained from D'_c by removing (D_c4). Note that then D''_c has incomplete (initial) information about *Petrol*. Then, the above plan Δ_1 is no longer a safe plan for G as there is no guarantee that the car will have petrol at the time T_1 when the *TurnOn* action is assumed to take place. A safe plan is now given by $\Delta_2 = \{\textit{TurnOn}\ \texttt{happens-at}\ T_1, \textit{Fill}\ \texttt{happens-at}\ T_2\}$ with $T_2 \prec T_1 \prec T_f$. In the context of D''_c, the original plan Δ_1 will be called a *weak plan.* A weak plan is a set of h-propositions such that the extension it forms of the given domain description might not entail the given goal, but there is at least one model of the augmented domain description in which the goal holds true. A weak plan depends upon a set of assumptions, in the form of t-propositions, such that, if these assumptions were true (or could be made true) then the weak plan would be (or would become) a safe plan. In the example above Δ_1 is weak as it depends on the set of assumptions $A = \{\textit{Petrol}\ \texttt{holds-at}\ T_1\}$. Δ_2 is obtained from Δ_1 by adding the additional action *Fill* `happens-at` T_2, ensuring that A is entailed by $D''_c \cup \Delta_2$.

Definition 4 *Given a planning domain $\langle D, P\rangle$ and a goal G, a* **weak plan for** G **in** D *is a set Δ of h-propositions s.t. there exists a model M of $D \cup \Delta$ where:*
- $M \models G$, *and*
- $M \models P$.

Δ *is* **conditional** *or* **depends on the set of assumptions** A *iff A is a set of t-propositions such that Δ is not a safe plan for G in D but it is a safe plan for G in $D \cup A$.*

Note that a safe plan is always a weak plan and that if a weak plan is not conditional on any assumptions then it is necessarily a safe plan.

Computing conditional weak plans will form the basis for computing safe plans in the $\mathcal{E}$-planner that we will develop in section . In general, if we have a weak plan for a given goal G, conditional on a set of assumptions A, then the original planning problem for G is reduced to the subsidiary problem of generating a plan for A. In effect, this process allows to actively fill in by further actions the incompleteness in the domain description.

However, in some cases we may have incomplete information on fluents that can not be affected by any further actions or at a time point (e.g. an initial time point) before which we cannot perform actions. In this paper we will concentrate on incompleteness of this kind, and we will study how to appropriately generate safe plans from weak plans despite the luck of information.

Of course, this may not always be possible, but there are many interesting cases, such as the following "vaccine" domain D_v, where a safe plan exists:

$$\textit{InjectA}\ \texttt{initiates}\ \textit{Protected}\ \texttt{when}\ \{\textit{TypeO}\} \quad (D_v1)$$
$$\textit{InjectB}\ \texttt{initiates}\ \textit{Protected}\ \texttt{when}\ \{\neg\textit{TypeO}\} \quad (D_v2)$$

Here, the fluent *TypeO* cannot be affected by any of action (we cannot change the blood type) and although its truth value is not known (we have incomplete information on this) we can still generate a safe plan for the goal, $G = \textit{Protected}\ \texttt{holds-at}\ T_f$, by performing both actions *InjectA* and *InjectB* before time T_f.

An Argumentation Formulation of $\mathcal{E}$

Argumentation has recently proved to be a unifying mechanism for most existing non-monotonic formalisms (Bondarenko *et al.* 1997; Dung 1995). In (Kakas, Miller, & Toni 1999), we have adapted the *LPwNF* (Dimopoulos & Kakas 1995) argumentation framework to provide an equivalent reformulation of the original Language $\mathcal{E}$ presented in section and to develop a proof theory for computing entailment of t-propositions in domain descriptions. This will form the computational basis for our $\mathcal{E}$-Planner. In this section, we give a brief review of the argumentation formulation for $\mathcal{E}$ concentrating on the methods and results that would be needed for the $\mathcal{E}$-Planner.

Let a *monotonic logic* be a pair $(\mathcal{L}, \vdash)$ consisting of a formal language $\mathcal{L}$ (equipped with a negation operator $\neg$) and a monotonic derivability notion $\vdash$ between sentences of the formal language. Then, an abstract *argumentation program*, relative to $(\mathcal{L}, \vdash))$, is a quadruple $(B, \mathcal{A}, \mathcal{A}', <)$ consisting of

- a *background theory* B, i.e. a (possibly empty) set of sentences in $\mathcal{L}$,
- an *argumentation theory* $\mathcal{A}$, i.e. a set of sentences in $\mathcal{L}$ (the *argument rules*),
- an *argument base* $\mathcal{A}' \subseteq \mathcal{A}$, and
- a *priority relation*, $<$ on the ground instances of the argument rules, where $\phi < \psi$ means that ϕ has lower priority than ψ.

Intuitively, any subset of the argument base can serve as a non-monotonic extension of the (monotonic) background theory, if this extension satisfies some requirements. The sentences in the background theory can be seen as non-defeasible argument rules which must belong to any extension. One possible requirement that extensions of the background theory must satisfy is that they are *admissible*, namely that they are:

- *non-self-attacking* and
- able to *counterattack* any (set of) argument rules *attacking* it.

Informally, a set of argument rules from $\mathcal{A}$ *attacks* another such set if the two sets are in conflict, by deriving in the underlying logic complimentary literals λ and $\neg\lambda$, respectively, and the subset of the attacking set (minimally) responsible for the derivation of λ is *not overall lower in priority* than the subset of the attacked set (minimally) responsible for the derivation of $\neg\lambda$. A set of rules A is of lower priority than another set B if it has a rule of lower priority than some rule in B and does not contain any rule of higher priority than some rule in B.

Then any given sentence σ of $\mathcal{L}$ is a *credulous* (*sceptical*, resp.) non-monotonic consequence of an argumentation program iff $B \cup \Delta \vdash \sigma$ for *some* (*all*, resp.) maximally admissible extension(s) Δ of the program.

A domain description D without t-propositions can be translated into an argumentation program $(B(D), \mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\mathcal{E}}', <_{\mathcal{E}})$, referred to as $P_{\mathcal{E}}(D)$, such that there is a one-to-one correspondance between:

i) models of D and maximally admissible sets of arguments of $P_{\mathcal{E}}(D)$;

ii) entailment in $\mathcal{E}$ and sceptical non-monotonic consequences of $P_{\mathcal{E}}(D)$.

These equivalence results continue to hold when D contains t-propositions or r-propositions by simply considering only the admissible sets that confirm the truth of all such propositions in D.

The basic elements of the translation of domain descriptions D into argumentation programs $P_{\mathcal{E}}(D)$ are as follows. All individual h- and c-proposition translations as well as the relationships between time-points are included in the background theory $B(D)$, so that for all time-points T, T' and action constants A,

- $B(D) \vdash T \prec T'$ iff $T \prec T'$
- $B(D) \vdash \mathit{HappensAt}(A, T)$ iff A `happens-at` T is in D,
- for each c-proposition
  A `initiates` F `when` $\{L_1, \ldots, L_n\}$ in D
  (resp. A `terminates` F `when` $\{L_1, \ldots, L_n\}$),
  $B(D)$ contains the rule
  $\mathit{Initiation}(F, t) \leftarrow \mathit{HappensAt}(A, t), \Lambda(L_1), \ldots, \Lambda(L_n)$
  ($\mathit{Termination}(F, t) \leftarrow \mathit{HappensAt}(A, t), \ldots, \Lambda(L_n)$ resp.), where $\Lambda(L_i) = (\neg)\mathit{HoldsAt}(F_i, t)$ if $L_i = (\neg)F_i$, for some fluent constant F_i.

As an example, consider the domain description D_c in section . Then, $B(D_c)$ contains the fact $\mathit{HappensAt}(\mathit{TurnOn}, 5)$ and $B(D_c)$ contains the rules

$$\mathit{Initiation}(\mathit{Running}, t) \leftarrow \mathit{HappensAt}(\mathit{TurnOn}, t), \mathit{HoldsAt}(\mathit{Petrol}, t)$$

$$\mathit{Termination}(\mathit{Petrol}, t) \leftarrow \mathit{HappensAt}(\mathit{Empty}, t).$$

The remaining components of $P_{\mathcal{E}}(D)$ are independent of the chosen domain D:

- $\mathcal{A}_{\mathcal{E}}$ consists of

Generation rules:

$$\mathit{HoldsAt}(f, t_2) \leftarrow \mathit{Initiation}(f, t_1), t_1 \prec t_2 \quad (PG[f, t_2; t_1])$$

$$\neg\mathit{HoldsAt}(f, t_2) \leftarrow \mathit{Termination}(f, t_1), t_1 \prec t_2 \quad (NG[f, t_2; t_1])$$

Persistence rules:

$$\mathit{HoldsAt}(f, t_2) \leftarrow \mathit{HoldsAt}(f, t_1), t_1 \prec t_2 \quad (PP[f, t_2; t_1])$$

$$\neg\mathit{HoldsAt}(f, t_2) \leftarrow \neg\mathit{HoldsAt}(f, t_1), t_1 \prec t_2 \quad (NP[f, t_2; t_1])$$

Assumptions:

$$\mathit{HoldsAt}(f, t) \quad (PA[f, t])$$

$$\neg\mathit{HoldsAt}(f, t) \quad (NA[f, t])$$

- $\mathcal{A}_{\mathcal{E}}'$ consists of all the generation rules and assumptions only.
- $<_{\mathcal{E}}$ is such that the effects of later events take priority over the effects of earlier ones. Thus persistence rules have lower priority than "conflicting" and "later" generation rules, and "earlier" generation rules have lower priority than "conflicting" and "later" generation rules. In addition, assumptions have lower priority than "conflicting" generation rules. For example, given the vocabulary of D_c in section , $PA[\mathit{Running}, 5] <_{\mathcal{E}} NG[\mathit{Running}, 5; 3]$ and $NG[\mathit{Running}, 7; 3] <_{\mathcal{E}} PG[\mathit{Running}, 7; 5]$.

Given this translation of the language $\mathcal{E}$ a proof theory can be developed by adapting the abstract, argumentation-based computational framework in (Kakas & Toni 1999) to the argumentation programs $P_{\mathcal{E}}(D)$. The resulting proof theory is defined in terms of *derivations of trees*, whose nodes are sets of arguments in $\mathcal{A}_{\mathcal{E}}$ attacking the arguments in their parent nodes. Let S_0 be a (non-self-attacking) set of arguments in $\mathcal{A}_{\mathcal{E}}'$ such that $B(D) \cup S_0 \vdash (\neg)\mathit{HoldsAt}(F, T)$, for some literal $(\neg)$ F `holds-at` T that we want to prove to be entailed by D (S_0 can be easily built by backward reasoning). Then, two kinds of derivations are defined:

- *Successful derivations*, building, from a tree consisting only of the root S_0, a tree whose root S is an admissible subset of $\mathcal{A}_{\mathcal{E}}'$ such that $S \supseteq S_0$.
- *Finately failed derivations*, guaranteeing the absence of any admissible set of arguments containing S_0.

Then, the given literal is entailed by D if there exists a successful derivation with inital tree consisting only of the root S_0 and, for every set S_0' of argument rules in $\mathcal{A}_{\mathcal{E}}'$ such that $B(D) \cup S_0'$ derives (in $\vdash$ the complement of the given literal, every derivation for S_0' is finitely failed.

This method is extended in the obvious way to handle conjunctions of literals rather than individual literals by choosing S_0 and S_0' appropriately. Also when a domain D contains t-propositions we simply conjoin these to the literals S_0 and S_0'. A similar extension of requiring that all the r-propositions are satisfied together with the goal at hand is applied for the domains containing such ramification statements.

The details of the derivations are not needed for the purposes of this paper. Informally, both kinds of derivation incrementally consider all attacks (sets of arguments in $\mathcal{A}_{\mathcal{E}}$) against S_0 and, whenever the root does not itself counterattack one of its attacks, a new a new set of arguments in $\mathcal{A}_{\mathcal{E}}'$ that can attack back this attack is generated and added to the root. Then, the process is repeated, until every attack has been counterattacked successfully (successful derivation) by the extended root or until some attack cannot be possibly counterattacked by any extension of the root (finitely failed derivations) During this process, the counterattacks are chosen in such a way that they do not attack the root. For example, for the domain D_c in section , given $S_0 = \{PG[Running, 7; 5], PA[Petrol, 5]\}$, monotonically deriving $HoldsAt(Running, 7)$, a successful derivation is constructed as follows:

$$\begin{array}{ccc} S_0 & S_0 & S_0 \\ & | & | \\ & K & K \\ & & | \\ & & S_0 \end{array}$$

S_0 is attacked by $K = \{NA[Petrol, 5]\}$, trivially counterattacked by S_0 itself. Thus, in this simple example, no extension of the root is required.

The $\mathcal{E}$-Planner

The argumentation-based techniques discussed in the previous section can be directly extended to compute plans for goals. (In the sequel, we will sometimes mix the original Language $\mathcal{E}$ formulation of problems and their corresponding formulation in the argumentation reformulation.) First, given a goal G, in order to derive the (translation ($\Lambda(G)$ of the) goal in the underlying monotonic logic, a preliminary step needs to compute not only a set of argument rules S_0, but (possibly) also a set **action facts** $\Delta_0 \subseteq \mathcal{H}$ where

$\mathcal{H} = \{HappensAt(F, T)|\ F$ is a fluent constant and T is a time-point$\}$

Δ_0 can be seen as a preliminary plan for the goal, that needs to be extended first to a weak plan and then to a safe plan. Every time a new action fact is added to a plan, any preconditions of the action need to be checked and, possibly, enforced, by adding further action facts. The computation of safe plans from weak ones requires blocking, if needed, any (weak) plan for the complement of any literal in the goal.

The following is a high-level definition of the $\mathcal{E}$-Planner in terms of the argumentation-based reformulation of the $\mathcal{E}$-language:

Definition 5 *Given a planning domain $\langle D, P\rangle$ and a goal G, an* **$\mathcal{E}$-plan for** *G is a set h-propositions $\Delta^{\mathcal{E}}$ such that $\Delta^{\mathcal{E}} = \{A$ `happens-at` $T | HappensAt(A, T) \in \Delta\}$, where $\Delta \subseteq \mathcal{H}$ is derived as follows:*

1) Find a set of arguments $S_0 \subseteq \mathcal{A}_{\mathcal{E}}'$ and a set of action facts $\Delta_0 \subseteq \mathcal{H}$ such that $B(D) \cup \Delta_0 \cup S_0 \vdash G$;

2) Construct a set of arguments $S \subseteq \mathcal{A}_{\mathcal{E}}'$ and a set of action facts $\Delta_w \subseteq \mathcal{H}$ such that (i) $S_0 \subseteq S$ and $\Delta_0 \subseteq \Delta_w$, and (ii) S is an admissible set of arguments wrt the augmented argumentation program $P_{\mathcal{E}}(D' \cup \Delta_w) = (B(D') \cup \Delta_w, \mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\mathcal{E}}', <_{\mathcal{E}})$, where $D' = D \cup \{\Lambda(C(T)) | A$ `needs` $C \in P, HappensAt(A, T) \in \Delta_w\}$.

3) If every assumption in S is a sceptical non-monotonic consequence of the augmented argumentation program $P_{\mathcal{E}}(D' \cup \Delta_w)$ then $\Delta = \Delta_w$.

4) Otherwise, Δ is a set of action facts such that $\Delta_w \subset \Delta$ and:

4.1) For every set of arguments $R \subseteq \mathcal{A}_{\mathcal{E}}'$ such that $B(D) \cup \Delta \cup R \vdash \neg G$, where $\neg G$ stands for the complement of some literal in G, there exists no $R' \subseteq \mathcal{A}_{\mathcal{E}}'$ such that $R \subseteq R'$ and R' is admissible wrt the augmented argumentation program $P_{\mathcal{E}}(D'' \cup \Delta) = (B(D'') \cup \Delta, \mathcal{A}_{\mathcal{E}}, \mathcal{A}_{\mathcal{E}}', <_{\mathcal{E}})$, where $D'' = D \cup \{\Lambda(C(T)) | A$ `needs` $C \in P, HappensAt(A, T) \in \Delta\}$.

4.2) There exists a set $S' \subseteq \mathcal{A}_{\mathcal{E}}'$ such that $S \subseteq S'$ and S' is admissible wrt $P_{\mathcal{E}}(D'' \cup \Delta)$.

In the first two steps, the $\mathcal{E}$-Planner computes a weak plan for the given goal. If this does not depend on any assumptions (step 3) then it is a safe plan, as no plan for the complement of the goal is possible. Otherwise (step 4), the planner attempts to extend the weak plan in order to block the derivation of the complement, $\neg G$, of the goal. In order to do so, it considers each possible set of arguments, R, which would derive $\neg G$ (in the augmented background theory) and extends the plan so that R can not belong to any admissible set of the resulting theory. A successful completion of step 4 means that the weak plan Δ_w has been rendered into a safe plan Δ.

The correctness of the planner is a direct consequence of the correctness of the argumentation proof theory on which it is based.

Theorem 1 *Given a planning domain $\langle D, P\rangle$ and a goal G, let Δ_w be the set of action facts computed at step 2 in definition 5. Then, the set*

$\Delta_w^{\mathcal{E}}=\{A$ `happens-at` $T | Happens(A,T) \in \Delta_w\}$
is a weak plan for G.

Proof: As every admissible set of arguments is contained in some maximally admissible set (Kakas, Miller, & Toni 1999) and, as discussed in section , every maximally admissible extension of the argumentation-based reformulation of a domain description in $\mathcal{E}$ corresponds to a model of the original domain, S computed at step 2 in definition 5 corresponds to a model M of $D' \cup \Delta_w^{\mathcal{E}}$ entailing G. Because of the way t-propositions in domains are handled, as additional conjuncts in goals, this implies that M satisfies all preconditions of actions in $\Delta_w^{\mathcal{E}}$. Thus, M is a model of $D \cup \Delta_w^{\mathcal{E}}$ such that $M \models G$ and $M \models P$ and the theorem is proven.

The following theorem can be proven in a similar way:

Theorem 2 *Given a planning domain $\langle D, P\rangle$ and a goal G, let $\Delta^{\mathcal{E}}$ be an $\mathcal{E}$-plan for G. Then, $\Delta^{\mathcal{E}}$ is a safe plan for G.*

The high-level definition of the $\mathcal{E}$-Planner given above in definition 5 can be mapped onto a more concrete planner by suitably extending the argumentation-based proof theory proposed in (Kakas, Miller, & Toni 1999). Δ_0 can be computed directly while computing S_0, by an **abductive process** which reasons backwards with the sentences in the background theory. Also, one needs to define suitable:

- **extended successful derivations**, for computing incrementally Δ_w from Δ_0 at step 2 and the final Δ at step 4.2 from the extension of Δ_w computed at step 4.1;
- **extended finitely failed derivations**, for computing incrementally the required extension of Δ_w at step 4.1.

As the original derivations, the extended ones incrementally consider all attacks against the root of the trees they build and augment the root so that all such attacks are counterattacked, until every attack has been counterattacked (successful derivations) or some attack cannot be counterattacked (failed derivations). Again, all nodes of trees are sets of arguments.

In addition, both new kinds of derivation are integrated with abduction to generate action facts so that success and failure are guaranteed, respectively. The action facts are chosen to allow for counterattacks to exist (successful derivations) or for additional attacks to be generated (failed derivations). Thus, extended successful derivations return both a set of argument rules in $\mathcal{A}_{\mathcal{E}}'$ and a set of action facts, wherever extended failed derivations just return a set of action facts (the ones needed to guarantee failure).

Both kinds of derivations need to add to the given background theory (domain) the t-propositions that are preconditions of any abduced action. By the way t-propositions are handled, this amounts to extending dynamically the given goal to prove or disprove, respectively.

Finally, both kinds of derivation require the use of **suspended nodes**, namely that could potentially attack or counterattack their parent node if some action facts were part of the domain. These nodes become actual attacks and counterattacks if and when the action facts are added to the accumulated set. If, at the end of the derivations, these action facts are not added, then suspended nodes remain so and do not affect the outcome of the derivations.

Let us illustrate the intended behaviour of the extended derivations with a simple example. Consider the simple "car engine" domain D_c' in section , and let $G = HoldsAt(Running, T_f)$ for some fixed final time T_f. We will show how the safe $\Delta_w = \{HappensAt(TurnOn, T_1)\}$, with $T_1 \prec T_f$, is computed.

1) $S_0 = \{PG[Running, T_f; T_1], PA[Petrol, T_1]\}$ and $\Delta_w = HappensAt(TurnOn, T_1)$, with $T_1 \prec T_f$.

2) The only possible attack against S_0 is $\{NA[Petrol, T_1]\}$, which is trivially counterattacked by S_0 itself. Thus S_0 is admissible.

3) Let us examine the only assumption $PA[Petrol, T_1]$ in S_0, and try to prove that it holds in all admissible extensions of the given domain extended by Δ_w. Consider the complement $NA[Petrol, T_1]$ of the assumption, and let us prove that it holds in no admissible extension. This can be achieved by an ordinary finitely failed derivation (without abducing any additional action fact in order to do so), as there is an attack, $\{PP[Petrol, T_1; 1], PA[Petrol, 1]\}$, against the above complement, which cannot be counterattacked.

Thus, Δ_w is a safe plan for G. Consider now the domain $D_c'' = D_c' - \{(D_c4)\}$. Then, step 3 above fails to prove that the given assumption is a sceptical non-monotonic consequence of the augmented domain, and thus Δ_w is just a weak plan.

4.1) $\neg G = \neg HoldsAt(Running, T_f)$ is derivable via $R = \{NA[Running, T_f]\}$. This is attacked by $\{PG[Running, T_f; T_1], PA[Petrol, T_1]\}$, which is counterattacked by $\{NA[Petrol, T_1]\}$, which, if added to R, would provide an admissible extension in which $\neg G$ holds. An extended finitely failed derivation can be constructed to prevent this as follows: the extended root $R \cup \{NA[Petrol, T_1]\}$ is attacked by $\{PG[Petrol, T_1; T_2]\}$ if Δ_w is augmented to give $\Delta = \Delta_w \cup \{HappensAt(Fill, T_2)\}$, with $T_2 \prec T_1$. As the new attack cannot be counterattacked, the derivation fails.

Thus, Δ is a safe plan for G (for simplicity we omit step 4.2 here).

Note that the method outlined above relies upon the explicit treatment of non-ground arithmetical constraints over time-points (see (Kakas, Michael, & Mourlas 1998)).

Incomplete planning problems

In this section we will illustrate through a series of examples the ability of the $\mathcal{E} - Planner$ to produce safe plans for incompletely specified problems. In particular, we will consider problems where the incompleteness on some of the fluents is such that it cannot be affected by actions and hence our knowledge of them cannot be (suitably) completed by adding action facts to plans.

Let us consider again the "vaccine" domain D_v at the end of section , and the goal $G = \{Protected$ `holds-at` $T_f\}$, for some final time T_f. We will show how, in this example, the $\mathcal{E}$-planner reasons correctly with the "excluded middle rule" to produce a safe plan for G, despite the fact that it is not known whether the blood if of $Type0$ or not. A weak plan for the goal is given by $\Delta_w = \{HappensAt(InjectA, T_1)\}$ with $T_1 \prec T_f$. Indeed: given $S_0 = \{PG[Protected, T_f; T_1], PA[Type0, T_1]\}$, $B(D_v) \cup S_0 \cup \Delta_w \vdash G$, S_0 is admissible (steps 1 and 2). The plan Δ_w is conditional on the set of assumptions $\{PA[Type0, T_1]\}$ (step 3). Note that there is no action that can affect the fluent $Type0$, so Δ_w cannot be extended so that it can derive the assumption. Nevertheless, we can extend Δ_w to a safe plan Δ by constructing failed derivations for $\neg G = \{\neg HoldsAt(Protected, T_f)\}$, as illustrated below (step 4.1).

The only way to derive $\neg G$ is by means of the set of arguments $R_1 = \{NA[Protected, T_f]\}$. This is attacked by S_0 itself, which can be counterattacked (only) if the root R_1 is extended via the assumption $NA[Type0, T_1]$.

The initial root R_1 and thus the extended root R_2 are attacked by the set of arguments $\{PG[Protected, T_f; T_2], NA[Type0, T_2]\}$ with $T_2 \prec T_f$, provided we add to Δ_w the action fact $HappensAt(InjectB, T_2)$ to give a new plan $\Delta = \{HappensAt(InjectA, T_1), HappensAt(InjectB, T_2)\}$. In order to successfully counterattack the new attack we need to add further to the root the argument $PA[Type0, T_2]$, obtaining a new root R_3.

R_3 is (newly) attacked by $\{PP[Type0, T_1; T_2], PA[Type0, T_2]\}$, if $T_2 \prec T_1$, and by $\{NP[Type0, T_2; T_1], NA[Type0, T_1]\}$, if $T_1 \prec T_2$. (Note also that necessarily $T_1 \neq T_2$ as otherwise R_3 would attack itself.) These attacks can only be counterattacked via one generation rule for $\neg HoldsAt(Type0, T_1)$ and one for $HoldsAt(Type0, T_2)$, respectively. But no such generation rules are possible.

This concludes the construction of the only required finitely failed extended derivation for $\neg G$. (Again, we omit step 4.2.) The computed $\mathcal{E}$-plan Δ is indeed a safe plan for G. Note that no p-propositions are present in this example and thus no extended domain is generated.

The following example illustrate the use of observations (in the form of t-propositions) to provide some (partial) implicit information on the domain. Consider the domain D_i:

InjectC `initiates` *Protected* `when` $\{TypeA\}$ (D_i1)
InjectD `initiates` *Protected* `when` $\{Weak\}$ (D_i2)
Bite `initiates` *Infected* `when` $\{TypeA\}$ (D_i3)
Expose `initiates` *Infected* `when` $\{Weak\}$ (D_i4)
$\neg Infected$ `holds-at` 1 (D_i5)
Infected `holds-at` 4 (D_i6)

and the goal $G = Protected$ `holds-at` T_f, for some final time $4 \prec T_f$. The fluents *TypeA* and *Weak* are incompletely specified. but the observations (t-propositions) essentially give indirectly the information that either *TypeA* or *Weak* must hold. This then allows, similarly to the previous example, to generate a safe plan for G by applying both actions *InjectC* and *InjectD*.

A weak plan $\Delta_w = \{HappensAt(InjectC, T_1)\}$ with $T_1 \prec T_f$ is first generated (steps 1 and 2), conditional on the assumption set $\{PA[TypeA, T_1]\}$ (step 3). Then, step 4.1 generates $R = \{NA[Protected, T_f]\}$, proving $\neg G$. R is attacked by $\{PG[Protected, T_f; T_1], PA[TypeA, T_1]\}$ and can only be defended if R is extended to $R' = R \cup \{NA[TypeA, T_1]\}$, which is admissible. However, R' needs to be extended to confirm the t-propositions. To confirm (D_i5), the assumption $NA[Infected, 1]$ needs to be added to R'. Moreover, to confirm (D_i6), we need to add either $R_1 = \{PG[Infected, T_f; T_2], PA[TypeA, T_2]\}$ or $R_2 = \{PG[Infected, T_f; T_2], PA[Weak, T_2]\}$, with $T_2 \prec 4$. But adding the fist such set would render the resulting set non-admissible (as both $PA[TypeA, T_2]$ and $NA[TypeA, T_1]$ belong to it). Hence, the only viable extension is $R'' = R' \cup R_2 \cup \{NA[Infected, 1]\}$. This set is admissible (if we also abduce the action $HappensAt(Expose, T_2)$). To prevent that, we find an attack that cannot be counterattacked successfully: $\{PG[Protected, T_f; T_3], PA[Weak, T_3]\}$, $T_3 \prec T_f$, extending Δ_w to $\Delta = \Delta_w \cup \{HappensAt(InjectD, T_3)\}$. This attack can only be counterattacked by adding the assumption $NA[Weak, T_3]$, rendering the root self-attacking (as $PA[Weak, T_2]$ belongs to it). Thus, Δ is a safe plan.

The next example shows how the $\mathcal{E}$-planner exploits ramification information to generate safe plans for incompletely specified problems. Let D_r be:

InjectB `initiates` *Protected* `when` $\{\neg TypeO\}$ (D_r1)
InjectE `initiates` *Protected* `when` $\{Strong\}$ (D_r2)
Strong `whenever` $\{TypeO\}$ (D_r3)

and $G = Protected$ `holds-at` T_f, for some final time $4 \prec T_f$. The fluents *TypeO* and *Strong* are incompletely specified. The ramification statement requires that either $\neg TypeO$ or *Strong* must hold (at any time). Then, similarly to the above examples, the $\mathcal{E} - planner$ can generate the safe plan $\{HappensAt(InjectB, T_1), HappensAt(InjectE, T_2)\}$, with $T_1, T_2 \prec T_f$. During the computation of this plan, to render the only proof of $\neg G$ non-admissible we generate, in addition to the attack given by the weak plan $\{HappensAt(InjectB, T_1)\}$, an extra attack by adding the action $HappensAt(InjectE, T_2)$. The first attack can only be counterattacked by $\{PA[Type0, T_1]\}$ and the second only by $\{NA[Strong, T_2]\}$. As these assumptions persist, it is not possible to satisfy the ramification statement at any time between T_1 and T_2. Hence there is no admissible extensions that can prove $\neg G$ and the plan is safe.

Conclusions

We have shown how we can formulate planning within the framework of the Language $\mathcal{E}$ and have used the argumentation reformulation of this framework to define a planner that is able to solve problems with incomplete information.

A planner with similar aims has been defined in (Finzi, Pirri, & Reiter 1999). Both this planner and our $\mathcal{E}$-planner regress to a set of assumptions which, when entailed by the incomplete theory, guarantees the plan to be safe. However, (Finzi, Pirri, & Reiter 1999) uses a classical theorem prover to check explicitly this entailment at the initial situation (in general, the required entailment is the non-monotonic entailment of the action framework in which the planning problems are formulated). Instead, in the $\mathcal{E}$-planner the incompleteness, and hence the assumptions to which one regresses, need not refer to the initial state only. Moreover, the $\mathcal{E}$-planner uses these assumptions in the computation of the initial possibly weak plan and then to help in the extension of this to a safe plan. We are studying other planning algorithms (within the same argumentation formulation of the language $\mathcal{E}$) which use more actively the assumptions on which weak plans are conditional. One such possibility is to try to extend the plan so as to re-prove the goal but now assuming a-priori the contrary of these assumptions. The search space of this type of planning algorithm is different and comparative studies of effeciency can be made.

(Smith & Weld 1998) introduce the notion of *conformant planning* for problems with incomplete information about the initial state and for problems where the outcome of actions may be uncertain. Our safe plans correspond to conformant plans for problems of the first type. The emphasis of this work is on the development of an efficient extension of Graphplan to compute conformant plans, by first considering all possible worlds and, in each world, all possible plans, and then extracting a conformant plan by considering the interactions between these different plans and worlds.

(Giunchiglia 2000) considers the problem of planning with incomplete information on the initial state within the action language $\mathcal{C}$. This is an expessive language that allows concurrent and non-deterministic actions together with ramification and qualification statements. Our safe plans correspond to the notion of *valid plans* which in turn are conformant plans. To find a valid plan, the $\mathcal{C}$-Planner generates a possible plan and then tests, using a SAT solver, whether the generated plan can be executed in all the possible models. Possible plans can be seen as weak plans, but we allow in the "testing phase" for the dynamic expansion of the weak plan into a safe plan.

A general difference with both (Smith & Weld 1998; Giunchiglia 2000) is that the $\mathcal{E}$-planner is goal-oriented, with an active search for actions both for the satisfaction of the goal and for ensuring that the generated plan is executable in any of the many possible worlds for the problem.

Another difference is at the level of expressivenes, in that we allow observations (not only at an initial state) and can exploit indirect information given by them to help handle the incompleteness.

We are currently developing an implementation of the $\mathcal{E}$-planner based on an earlier implmentation of the language $\mathcal{E}$ and aim to carry out experiments with standard planning domains. In this initial phase of our study we have not considered efficiency issues, concentrating specifically on the problem of planning under incompletness. In future work we need to address these issues by studying the problem of effective search in the space of solutions. One way to do this is to consider the integration of constraint solving in the planner as in constraint logic programming and its extension with abduction (Kakas, Michael, & Mourlas 1998; Kowalski, Toni, & Wetzel 1998).

We are considering several extensions of the $\mathcal{E}$-planner to allow for more general plans e.g. containing non-deterministic actions (or actions with uncertain effects). These extensions would require corresponding extensions of the expressiveness of the Language $\mathcal{E}$. Also, the extension of the $\mathcal{E}$-planner to accommodate sensing, in the form of accepting further observations (t-propositions) in the problem description, is a natural problem for future work.

Acknowledgements

This research has been partially supported by the EC Keep-In-Touch Project "Computational Logic for Flexible Solutions to Applications". The third author has been supported by the UK EPSRC Project "Logic-based multi-agent systems".

References

Bondarenko, A.; Dung, P. M.; Kowalski, R. A.; and Toni, F. 1997. An abstract, argumentation-theoretic framework for default reasoning. *Journal of Artificial Inelligence* 93(1-2):63–101.

Denecker, M.; Missiaen, L.; and Bruynooghe, M. 1992. Temporal reasoning with abductive event calculus. In *Proceedings of ECAI'92.*

Dimopoulos, Y., and Kakas, A. 1995. Logic programming without negation as failure. In *Proceedings of ILPS'95, 369–383.*

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of ECP'97, Springer Verlag, 169–181.*

Dung, P. 1995. The acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming and n-person game. *Journal of Artificial Inelligence* 77:321–357.

Finzi, A.; Pirri, F.; and Reiter, R. 1999. Open world planning in the situation calculus. In *Technical Report, University of Toronto.*

Giunchiglia, E. 2000. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Proceedings of KR'2000.*

Kakas, A., and Miller, R. 1997a. Reasoning about actions, narratives and ramifications. In *J. of Electronic Transactions on A.I. 1(4), Linkoping University E. Press, http://www.ep.liu.se/ea/cis/1997/012/.*

Kakas, A., and Miller, R. 1997b. A simple declarative language for describing narratives with actions. In *JLP 31(1–3), 157–200.*

Kakas, A., and Toni, F. 1999. Computing argumentation in logic programming. In *JLC 9(4), 515–562, O.U.P.*

Kakas, A.; Michael, A.; and Mourlas, C. 1998. Aclp: a case for non-monotonic reasoning. In *Proceedings of NMR98*, 46–56.

Kakas, A.; Miller, R.; and Toni, F. 1999. An argumentation framework for reasoning about actions and change. In *LPNMR'99, 78–91, Springer Verlag.*

Kowalski, R. A.; Toni, F.; and Wetzel, G. 1998. Executing suspended logic programs. *Fundamenta Informaticae* 34(3):203–224.

Levesque, H. 1996. What is planning in the presence od sensing? In *Proceedings of AAAI'96*, 1139–1146.

Lifschitz, V. 1999. Answer set planning. In *Proceedings of ICLP'99*, 23–37.

Shanahan, M. 1997. Event calculus planning revisited. In *Proceedings of ECP'97, Springer Verlag, 390–402.*

Smith, D., and Weld, D. 1998. Conformant graphplan. In *Proceedings of AAAI'98.*