Planning partially for situated agents

Paolo Mancarella[1], Fariba Sadri[2], Giacomo Terreni[1], and Francesca Toni[2,1]

[1] University of Pisa, Pisa, Italy
email: {paolo,terreni,toni}@di.unipi.it
[2] Department of Computing, Imperial College London, UK
email: {fs,ft}@doc.ic.ac.uk

Abstract. In recent years, within the planning literature there has been a departure from approaches computing *total plans* for given goals, in favour of approaches computing *partial plans*. Total plans can be seen as (partially ordered) sets of actions which, if executed successfully, would *lead* to the achievement of the goals. Partial plans, instead, can be seen as (partially ordered) sets of actions which, if executed successfully, would *contribute* to the achievement of the goals, subject to the achievement of further *sub-goals*. Planning partially (namely computing partial plans for goals) is useful (or even necessary) for a number of reasons: (i) because the planning agent is resource-bounded, (ii) because the agent has incomplete and possibly incorrect knowledge of the environment in which it is situated, (iii) because this environment is highly dynamic. In this paper, we propose a framework to design situated agents capable of planning partially. The framework is based upon the specification of planning problems via an abductive variant of the event calculus.

1 Introduction

Conventional GOFAI planners and planning techniques (e.g. [1]) rely upon a number of assumptions: (i) that the planning agent can devote as many resources as required to the planning task, and thus it can keep on planning until a *total plan* for some given goals is obtained, (ii) that the knowledge of the agent is complete and correct at the planning time, and (iii) that the environment in which the agent is situated will not change between the planning time and the time of execution of the plan, and thus the plan will be directly executable, thus leading to achieving the goals it is meant to achieve. These assumptions are unrealistic in most cases where planning is used, e.g. when the planning agent is a robot in a dynamic physical environment.

A number of approaches have been proposed in the literature to cope with the limitations of GOFAI planners, starting from early work on hierarchical planning. In this paper, we present an approach to planning whereby the planning agent generates possibly *partial plans*, namely (partially ordered) sets of actions which, if executed successfully, would *contribute* to the achievement of the goals, subject to the achievement of further *sub-goals*. A partial plan, like a hierarchical plan, is obtained by decomposition of *top-level goals*. A partial plan consists of sub-goals, that still need to be planned for, and *actions*, that can be directly executed, subject to their *preconditions* holding. Preconditions are also part of partial plans, and they need planning for before the actions can be executed. Within our approach, the decomposition of top-level goals, sub-goals and

preconditions into total plans is interleaved with the observation of the environment in which the agent is situated, via a *sensing* capability of the agent. Sensed changes in the environment are assimilated within the planning knowledge base of the agent. Currently, this assimilation is done rather straightforwardly, by adding the sensed information to the planning knowledge base and, if inconsistent with it, by "dropping" (implicitly) the existing beliefs in this knowledge base that lead to the inconsistency. Thus, our approach relies upon full trust upon the sensing capability of the agent. Observations from the environment in turn might lead to the need to revise the currently held partial plan, because as a consequence of the observations the agent notices that some top-level goals, sub-goals or preconditions already hold, or that they need to be re-planned for, or that they will never hold.

We adopt a novel variant of the event calculus [10], based upon abduction, to represent the planning knowledge base of agents, which allows to perform partial planning and to assimilate observations from the environment (in the simple manner described above). We represent top-level goals, sub-goals, preconditions and actions in the language of the event calculus. We impose a *tree structure* over top-level goals, sub-goals, preconditions and actions to support the revision of partial plans after observations and because of the passage of time. We define the behaviour of the planning agent via a $sense - revise - plan - execute$ life-cycle, which relies upon *(state) transitions* (for sensing, revision, planning and action execution) and *selection functions* to select intelligently top-level goals, sub-goals and preconditions to plan for and actions to be executed. A variant of the approach described here has been used within KGP agents [7, 2] and realized within the prototype implementation $PROSOCS$ [19] of KGP agents.

The paper is organised as follows. In section 2 we give some background on abductive logic programming with constraints, since the event calculus-based planning knowledge base of agents we adopt is a theory in this framework. In section 3 we give the planning knowledge base. In section 4 we define our partial plans and the cycle of planning agents. In section 5 we define the individual transitions. In section 6 we define the selection functions. In section 7 we give a simple example. In section 8 we evaluate our approach against related work and conclude.

2 Background: abductive logic programming with constraints

We briefly recall the framework of Abductive Logic Programming (ALP) for knowledge representation and reasoning [8], which underlies our planning technique. An *abductive logic program* is a triple $\langle P, A, I\rangle$ where:

- P is a *normal logic program*, namely a set of rules (clauses) of the form $H \leftarrow L_1, \ldots, L_n$ with H atom, $L_1, \ldots, L_n$ literals, and $n \geq 0$. Literals can be positive, namely atoms, or negative, namely of the form $not\, B$, where B is an atom. The negation symbol not indicates *negation as failure*. All variables in H, L_i are implicitly universally quantified, with scope the entire rule. H is called the *head* and $L_1, \ldots\ L_n$ is called the *body* of a rule. If $n = 0$, then the rule is called a *fact*.
- A is a set of *abducible predicates* in the language of P, not occurring in the head of any clause of P (without loss of generality, see [8]). Atoms whose predicate is abducible are referred to as *abducible atoms* or simply *abducibles*.
- I is a set of *integrity constraints*, that is, a set of sentences in the language of P. All the integrity constraints in this paper will have the implicative form $L_1, \ldots, L_n \Rightarrow$

$A_1 \vee \ldots \vee A_m \quad (n \geq 0, m > 1)$ where L_i are literals[3], A_j are atoms (possibly the special atom $false$). All variables in the integrity constraints are implicitly universally quantified from the outside, except for variables occurring only in the *head* $A_1 \vee \ldots \vee A_m$, which are implicitly existentially quantified with scope the head. $L_1, \ldots, L_n$ is referred to as the *body*.

Given an abductive logic program $\langle P, A, I \rangle$ and a formula (*query/observation/goal*) Q, which is an (implicitly existentially quantified) conjunction of literals in the language of the abductive logic program, the purpose of abduction is to find a (possibly minimal) set of (ground) abducible atoms Δ which, together with P, "entails" (an appropriate ground instantiation of) Q, with respect to some notion of "entailment" that the language of P is equipped with, and such that this extension of P "satisfies" I (see [8] for possible notions of integrity constraint "satisfaction"). Here, the notion of "entailment" depends on the semantics associated with the logic program P (there are many different possible choices for such semantics [8]). More formally and concretely, given a query Q, a set of (ground) abducible atoms Δ, and a variable substitution θ for the variables in Q, the pair (Δ, θ) is a *(basic) abductive answer* for Q, with respect to an abductive logic program $\langle P, A, I \rangle$, iff $P \cup \Delta \models_{LP} Q\theta$, and $P \cup \Delta \models_{LP} I$, where $\models_{LP}$ is a chosen semantics for logic programming. In this paper, we will not commit to any such semantics.

The framework of ALP can be usefully extended to handle constraint predicates in the same way Constraint Logic Programming (CLP) [6] extends logic programming. This extension allows to deal with non-ground abducibles, needed to support our planning approach. The CLP framework is defined over a particular structure $\Re$ consisting of domain $D(\Re)$ and a set of constraint predicates which includes equality, together with an assignment of relations on $D(\Re)$ for each constraint predicate. The structure is equipped with a notion of $\Re$-satisfiability. In this paper, the constraint predicates will be $<, \leq, >, \leq, =, \neq$, but we will not commit to any concrete structure for their interpretation. Given a (set of) constraints C, $\models_{\Re} C$ will stand for C is $\Re$-satisfiable, and $\sigma \models_{\Re} C$, for some grounding σ of the variables of C over $D(\Re)$, will stand for C is $\Re$-satisfied via σ.

The rules of a constraint logic program P take the same form as the rules in conventional logic programming, but with constraints occurring in the body of rules. Similarly, P and I in an abductive logic program might have constraints in their bodies. The semantics of a logic program with constraints is obtained by combining the logic programming semantics $\models_{LP}$ and $\Re$-satisfiability [6]. Below, we will refer to such a combined semantics as $\models_{LP(\Re)}$.

The notion of basic abductive answer can be extended to incorporate constraint handling as follows. Given a query Q (possibly with constraints), a set Δ of (possibly non-ground) abducible atoms, and a set C of (possibly non-ground) constraints, the pair (Δ, C) is an *abductive answer with constraints* for Q, with respect to an abductive logic program with constraints $\langle P, A, I \rangle$, with the constraints interpreted on $\Re$, iff for all groundings σ for the variables in Q, Δ, C such that $\sigma \models_{\Re} C$ then, (i) $P \cup \Delta\sigma \models_{LP(\Re)} Q\sigma$, and (ii) $P \cup \Delta\sigma \models_{LP(\Re)} I$.

In the sequel, we will use the following extended notion of abductive answer. Given an abductive logic program (with constraints) $\langle P, A, I \rangle$, a query Q (with constraints), an

[3] If $n = 0$, then $L_1, \ldots, L_n$ represents the special atom $true$.

initial set of (possibly non-ground) abducible atoms Δ_0 and an initial set of (possibly non-ground) constraint atoms C_0, an *abductive answer* for Q, with respect to $\langle P, A, I\rangle$, Δ_0, C_0, is a pair (Δ, C) such that $\Delta \cap \Delta_0 = \{\}$, $C \cap C_0 = \{\}$, and $(\Delta \cup \Delta_0, C \cup C_0)$ is an abductive answer with constraints for Q, with respect to $\langle P, A, I\rangle$.
In abductive logic programming (with constraints), abductive answers are computed via *abductive proof procedures*, which typically extend SLD-resolution, providing the computational backbone underneath most logic programming systems, in order to check and enforce integrity constraint satisfaction, the generation of abducible atoms, and the satisfiability of constraint atoms (if any). There are a number of such procedures in the literature, e.g. CIFF [4, 3]. Any such (correct) procedure could be adopted to obtain a concrete planning system based upon our approach. Within KGP agents [7, 19] we have adopted CIFF to perform the planning tasks along the lines described in this paper.

3 Representing a planning domain

In our framework, a planning problem is specified within the framework of the event calculus (EC) for reasoning about actions, events and changes [10], in terms of an abductive logic program with constraints $KB_{plan} = \langle P_{plan}, A_{plan}, I_{plan}\rangle$ and an ordinary logic program KB_{pre}. The EC allows to represent a wide variety of phenomena, including operations with indirect effects, non-deterministic operations, and concurrent operations [15]. A number of abductive variants of the EC have been proposed to deal with planning problems. Here, we propose a novel variant KB_{plan}, somewhat inspired by the $\mathcal{E}$-language of [9], to allow situated agents to generate partial plans in a dynamic environment. In a nutshell, the conventional EC allows to write meta-logic programs which "talk" about object-level concepts of *fluents*, *operations*, and *time points*. We allow fluents to be positive, indicated e.g. as F, or negative, indicated e.g. as $\neg F$. Fluent literals will be indicated e.g. as L. The main meta-predicates of the formalism are: $holds\_at(L, T)$ (a fluent literal L holds at a time T), $clipped(T_1, F, T_2)$ (a fluent F is clipped - from holding to not holding - between a time T_1 and a time T_2), $declipped(T_1, F, T_2)$ (a fluent F is declipped - from not holding to holding - between a time T_1 and a time T_2), $initially(L)$ (a fluent literal L holds at the initial time, say time 0), $happens(O, T)$ (an operation/action O happens at a time T), $initiates(O, T, F)$ (a fluent F starts to hold after an operation O at time T) and $terminates(O, T, F)$ (a fluent F ceases to hold after an operation O at time T). Roughly speaking, in a planning setting the last two predicates represent the cause-effects links between operations and fluents in the modelled world. We will also use a meta-predicate $precondition(O, L)$ (the fluent literal L is one of the preconditions for the executability of the operation O). In our novel variant we also use $executed$ and $observed$ predicates to deal with dynamic environments and the $assume\_holds$ predicate to allow for partial planning.
We now give KB_{plan}. P_{plan} consists of domain-independent and domain-dependent rules. The basic *domain-independent rules*, adapted from the original EC, are:

$$holds\_at(F, T_2) \leftarrow happens(O, T_1), initiates(O, T_1, F), T_1 < T_2, \neg\, clipped(T_1, F, T_2)$$

$$holds\_at(\neg F, T_2) \leftarrow happens(O, T_1), terminates(O, T_1, F), T_1 < T_2, \neg\, declipped(T_1, F, T_2)$$

$$holds\_at(F, T) \leftarrow initially(F), 0 < T, \neg\, clipped(0, F, T)$$

$holds\_at(\neg F, T) \leftarrow initially(\neg F), 0 < T, \neg\, declipped(0, F, T)$
$clipped(T_1, F, T_2) \leftarrow happens(O,\, T), terminates(O, T, F), T_1 \leq T < T_2$
$declipped(T_1, F, T_2) \leftarrow happens(O,\, T), initiates(O, T, F), T_1 \leq T < T_2$

The *domain-dependent rules* define the $initiates$, $terminates$, and $initially$ predicates. We show a simple example for such rules within the *blocks-world* domain.

Example 1. The domain dependent rules for the $mv(X, Y)$ operation in the block world domain, whose effects are to move block X onto block Y, are the following:

$initiates(mv(X, Y), T, on(X, Y))$
$terminates(mv(X, Y), T, clear(Y))$
$terminates(mv(X, Y), T, on(X, Z)) \leftarrow holds\_at(on(X, Z), T), Y \neq Z$
$initiates(mv(X, Y), T, clear(Z)) \quad \leftarrow holds\_at(on(X, Z), T), Y \neq Z$

namely the $mv(X, Y)$ operation *initiates* block X to be on block Y and *terminates* Y being clear. Moreover, if block X was on a block Z, the operation mv *terminates* this relation and *initiates* block Z being clear. □

The conditions for the rules defining $initiates$ and $terminates$ can be seen as preconditions for the effects of the operation (e.g. mv in the earlier example) to be established. Conditions for the executability of operations are specified within KB_{pre}, which consists of a set of rules defining the predicate $precondition$.

Example 2. The preconditions for the executability of operation $mv(X, Y)$ are that both X and Y are clear, namely:

$precondition(mv(X, Y), clear(X)) \qquad precondition(mv(X, Y), clear(Y))$

□

In order to accommodate (partial) planning we will assume that the domain-independent part in P_{plan} also contains the rules:

$happens(O, T) \leftarrow assume\_happens(O, T)$
$holds\_at(L, T) \leftarrow assume\_holds(L, T)$

i.e. an operation can be made to happen and a fluent can be made to hold simply by assuming them, where $assume\_happens$ and $assume\_holds$ are the only predicates in A_{plan} in KB_{plan}. This supports partial planning as follows. We will see that actions in our specification amount to atoms in the abducible predicate $assume\_happens$: thus, abducing an atom in this predicate amounts to planning to execute the corresponding action. Moreover, as yet unplanned for, sub-goals in our specification of partial plans amount to atoms in the abducible predicate $assume\_holds(L, T)$: abducing an atom in this predicate indicates that further planning is needed for the corresponding sub-goal. I_{plan} in KB_{plan} contains the following domain-independent integrity constraints:

$holds\_at(F, T), holds\_at(\neg F, T) \Rightarrow false$
$assume\_happens(A, T), not\ executed(A, T), time\_now(T') \Rightarrow T > T'$

namely a fluent and its negation cannot hold at the same time and when assuming (planning) that an action will happen, we need to enforce it to be executable in the future.

As we will see in section 4, a concrete planning problem is influenced (amongst other things) by a *narrative* of events, which, unlike KB_{plan} and KB_{pre}, changes over the life-cycle of the agent. We refer to the agent's representation of this narrative as KB_0. We assume that KB_0 represents events via predicates $executed$ and $observed$, e.g., the KB_0 of an agent in the blocks-world domain with a and b as two blocks, might contain:

$executed(mv(a, b), 3) \quad observed(\neg on(b, a), 10) \quad observed(ag, mv(c, d), 3, 5)$

namely the agent has *executed* a $mv(a, b)$ operation at time 3, the agent has *observed* that $\neg on(b, a)$ holds at time 10 and the agent has observed at time 5 that another agent ag has moved block c onto block d at time 3. Observations are drawn, via specific sensing capabilities of agents, from the environment in which they are situated, and are recorded in KB_0, as are records of actions executed by the agent itself. To allow agents to draw conclusions, via the EC, from the contents of KB_0 the following *bridge rules* are also contained in the domain independent rules of P_{plan}:

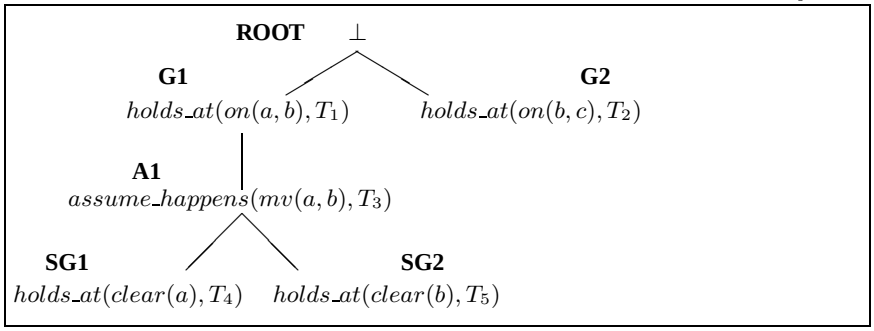
$$
\begin{array}{ll}
clipped(T_1, F, T_2) & \leftarrow observed(\neg F, T), T_1 \leq T < T_2 \\
declipped(T_1, F, T_2) & \leftarrow observed(F, T), T_1 \leq T < T_2 \\
holds\_at(F, T_2) & \leftarrow observed(F, T_1), T_1 \leq T_2, \neg\, clipped(T_1, F, T_2) \\
holds\_at(\neg F, T_2) & \leftarrow observed(\neg F, T_1), T_1 \leq T_2, \neg\, declipped(T_1, F, T_2) \\
happens(O, T) & \leftarrow executed(O, T) \\
happens(O, T) & \leftarrow observed(A, O, T', T)
\end{array}
$$

Note that we assume that the value of a fluent literal is changed according to observations only from the moment the observations are made, and actions by other agents have effects only from the time observations are made that they have been executed, rather than from the execution time itself. These choices are dictated by the rationale that observations can only have effects from the moment the planning agent makes them.

4 Representing planning problems and the life-cycle of agents

Given a planning domain and a set of (top-level) goals $Goals$ held by the agent, each of the form $holds\_at(L, T)$, we represent a *partial plan* for $Goals$ as a triple $\langle Strategy, Parent, TC\rangle$, where

- $Strategy$ is a set of *subgoals* and *preconditions*, each of the form $holds\_at(L, T)$, and of *actions*, each of the form $assume\_happens(L, T)$; each T of goals, subgoals, preconditions and actions is existentially quantified in the context of the goals and the partial plan; each such T is unique as we shall see in section 5; thus, such time variable uniquely identifies goals, subgoals, preconditions and actions;
- $Parent$ is a function from $Strategy$ to $Goals \cup Strategy$, inducing a *tree structure* over the $Goals$ and the $Strategy$; the root of this tree is the special symbol $\perp$, its children are all the goals in $Goals$, and the children of any other node in the tree is the set of all subgoals/preconditions/actions which are mapped, via $Parent$, onto the node; as we shall see in section 5, preconditions can only be children of actions, whereas subgoals and actions can be children of goals, subgoals or preconditions;
- TC is a set of *temporal constraints* over the times of goals, subgoals, preconditions and actions in $Strategy$, namely constraint atoms in the language of KB_{plan}.



Above we show a simple tree structure (where a Gn represents a goal, an SGn represents a subgoal and an An represents an action) for the blocks world domain, for the example given later in Section 7, to which we remand for details.

In the sequel, we will refer to any of goals, subgoals, preconditions and actions as *nodes*. Moreover, with an abuse of notation, we will represent nodes N in $Goals$ and $Strategy$ as pairs $\langle holds\_at(L,T), Pt\rangle$ and $\langle assume\_happens(O,T), Pt\rangle$, where $Pt = Parent(N)$, and we will omit mentioning $Parent$ in partial plans.
Given a planning domain, we represent a concrete planning problem, at a certain time τ (to be interpreted as the current time), via a notion of *state* defined below. Then, the planning process amounts to a sequence of such states, at incremental times, corresponding to the agent's life-cycle.

Definition 1. *An agent's* state *at time τ is a tuple $\langle KB_0, \Sigma, Goals, Strategy, TC\rangle$, where*

- *KB_0 is the recorded set of observations and executed operators (up until τ);*
- *Σ is the set of all bindings $T = X$, where T is the time variable associated with some action recorded as having been executed by the agent itself within KB_0, with the associated execution time X;*
- *$Goals$ is the set of (currently unachieved) goals, held by the agent at time τ;*
- *$\langle Strategy, TC\rangle$ is a partial plan for $Goals$, held by the agent at time τ;*

Below, by the tree corresponding to a state we mean the tree corresponding to the $Goals$ and $Strategy$ in the state, and to a node of the tree to indicate an element of $Strategy \cup Goals$, thus excluding $\bot$.
We now introduce the concepts of *initial state* and *final state*. An initial state is a state of the form $\langle\{\}, \{\}, Goals, \{\}, TC\rangle$, where TC are the given temporal constraints for $Goals$. The tree Tr_S corresponding to an initial state S is a two-level tree with root $\bot$ and all the goals in $Goals$ as the children of $\bot$. A final state can be either a *success state* or a *failure state*. A success state is a state of the form $\langle KB_0, \Sigma, \{\}, \{\}, TC\rangle$. A failure state is a state of the form: $\langle KB_0, \Sigma, \oslash, \{\}, TC\rangle$, where the symbol $\oslash$ indicates that there is no way to achieve *one* of the initial goals. [4]
In our framework, an agent which wants to plan in order to achieve its goals behaves according to a *life-cycle* which is an adaptation of the classical *sense - plan - execute* cycle. Concretely, such a life-cycle can be seen as the repetition of a sequence of steps

$$sense - revise - plan - execute$$

starting from an initial state until a final state is reached. In the next section we show the specification of the various steps, in the form of *state transitions*. Thus, the life-cycle of the planning agent can be equated to a sequence of states, each at a specific time τ. The corresponding tree varies during the life-cycle of the agent, by inserting and deleting nodes, as specified in the next section.
We will use the following notation. Given a state S, with its corresponding tree Tr_S:

- the set of *siblings* of a node $N \in Tr_S$ of the form $\langle \_, Pt\rangle$ is the set $Siblings(N, Tr_S) = \{N' \in Tr_S \mid N' = \langle \_, Pt\rangle\}$.

[4] This is an arbitrary decision, and we could have defined a failure state as one where there is no way to achieve all the goals, and a success state as one where at least one goal can be achieved.

– the set of *preconditions* of an action A of the form $\langle assume\_happens(O,T), Pt\rangle$ is the set $Pre(A, Tr_S) = \{P \in Tr_S \mid P = \langle \_, A\rangle\}$.

5 Transitions specification

Here we give the specification of the state transitions determining the life-cycle of the planning agent. We refer to these transitions as the *sensing transition*, the *planning transition*, the *execution transition*, and the *revision transition*. The planning and execution transitions take inputs that are computed via *selection functions*, defined in section 6.

5.1 Sensing Transition

Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, the application of a sensing transition at τ leads to a state $S' = \langle KB_0', \Sigma, Goals, Strategy, TC\rangle$, where KB_0' is obtained from KB_0 by adding any observations on fluent literals at τ and any observations at τ that an operation has been executed by another agent (at an earlier time). These observations are obtained by calling the *sensing capability* of the agent at time τ which we refer to as $\models^{\tau}_{Env}$, which accesses the environment of the agent.

Definition 2. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, if* $\models^{\tau}_{Env} l_1 \wedge \ldots \wedge l_n \wedge a_1 \wedge \ldots \wedge a_m$
where $n + m \geq 0$, each l_i is a fluent literal and each a_j is an operation o_j executed by agent ag_j at some earlier time τ_j, then the sensing transition *leads to a state $S' = \langle KB_0', \Sigma, Goals, Strategy, TC\rangle$ where:*

$$KB_0' = KB_0 \cup \{observed(l_1, \tau) \cup \ldots \cup observed(l_n, \tau)\} \cup \{observed(ag_1, o_1, \tau_1, \tau), \ldots, observed(ag_m, o_m, \tau_m, \tau)\}.$$

5.2 Planning Transition

The planning transition relies upon a *planning selection function* $SelP(S, \tau)$ which, given as input a state S at time τ returns a (single) goal, subgoal or precondition to be planned for. The extension whereby multiple goals, subgoals and preconditions are returned by the selection function is straightforward. In this section, we assume that such a selection function is given (a possible specification is provided in the next section).
We introduce the following useful notation which will be helpful in defining the planning transition. Let $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ be a state. Then:

– for any set $X \subseteq Goals \cup Strategy$, by $X(\Sigma)$ we denote the set obtained by applying to each element of X the instantiations provided by Σ;
– given a node $G \in Goals \cup Strategy$, by $Rest(G)$ we denote the set
$$Rest(G) = Strategy(\Sigma) \cup Goals(\Sigma) - G(\Sigma);$$
– given a node $N \in Goals \cup Strategy$, we denote by $\mathcal{A}(N)$ the *abducible version* of N, namely
$$\mathcal{A}(N) = \begin{cases} assume\_happens(O,T) & \text{if } N = \langle assume\_happens(O,T), \_\rangle \\ assume\_holds(L,T) & \text{if } N = \langle holds\_at(L,T), \_\rangle \end{cases}$$
This notation is lifted to any set X of nodes as usual, i.e. $\mathcal{A}(X) = \bigcup_{N \in X} \mathcal{A}(N)$.

Intuitively, given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC \rangle$, the planning transition builds a (partial) plan for a given goal, subgoal or precondition G in terms of an abductive answer, as defined in section 2, and updates the state accordingly. More precisely, an abductive answer is computed with respect to:

- the abductive logic program with constraints KB_{plan}, as defined in Section 3;
- the initial query Q given by G;
- the initial set of abducibles Δ_0 given by the abducible version of the current tree (except for G), namely $\mathcal{A}(Rest(G))$;
- the initial set of constraints C_0 given by the current set of constraints in the state, along with the instantiations in Σ, namely $TC \cup \Sigma$.

Once such abductive answer, say (Δ, C'), is obtained, the planning transition leads to a new state $S' = \langle KB_0, \Sigma, Goals, Strategy', TC' \rangle$ where $Strategy'$ is $Strategy$ augmented with the actions, goals and preconditions derived from Δ, and TC' is TC augmented with C' and with suitable equalities on the time variables of the preconditions of actions added to the state. We assume that the abducibles in Δ do not share time variables[5]. This is formalised in the next definition.

Definition 3. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC \rangle$ at a time τ and the node $G = SelP(S, \tau)$, let (Δ, C') be an abductive answer for the query G with respect to the abductive logic program (with constraints) KB_{plan}, and initial sets $\Delta_0 = \mathcal{A}(Rest(G))$ and $C_0 = TC \cup \Sigma$. Then, the* planning transition *leads to a state $S' = \langle KB_0, \Sigma, Goals, Strategy', TC' \rangle$ where $Strategy'$ and TC' are obtained by augmenting $Strategy$ and TC as follows:*

- *for each $assume\_holds(L, T) \in \Delta$, $\langle holds\_at(L, T), G \rangle$ is added in $Strategy'$*
- *for each $assume\_happens(O, T) \in \Delta$*
  - *$A = \langle happens(O, T), G \rangle$ is added in $Strategy'$, and*
  - *for each P such that $precondition(happens(O, T), P) \in KB_{pre}$, let T_p be a fresh time variable; then:*
    $\langle holds\_at(P, T_P), A \rangle$ is added in $Strategy'$, and
    $T_P = T$ is added in TC'
  - *C' is added in TC'*

Note that this transition enforces that preconditions of actions hold at the time of the execution of the actions, by adding such preconditions to $Strategy'$ so that they will need planning for. Note also that, when introducing preconditions, we need to make sure that their time variable is new, and relate this, within TC', to the time variable of the action whose preconditions we are enforcing.

5.3 Execution Transition

Similarly to the planning transition, the execution transition relies upon an *execution selection function* $SelE(S, \tau)$ which, given a state S and a time τ, returns a (single) action to be executed (a possible specification of this selection function is provided in the next section). The extension to the case of multiple actions is straightforward.

[5] Notice that this is not a restrictive assumption, since shared variables can be renamed and suitable equalities can be added to the constraints in C'.

Definition 4. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ and an action A of the form $\langle assume\_happens(O,T), Pt\rangle$ such that $A = SelE(S,\tau)$, then the* execution transition *leads to a state $S' = \langle KB_0', \Sigma', Goals, Strategy, TC\rangle$ where:*

- $KB_0' = KB_0 \cup \{executed(O,\tau)\}$
- $\Sigma' = \Sigma \cup \{T = \tau\}$

Note that we are implicitly assuming that actions are ground except for their time variable. The extension to deal with other variables in actions is straightforward.
Executed actions are eliminated from states by the revision transition, presented next.

5.4 Revision Transition

To specify the revision transition we need to introduce some useful concepts. A node is said to be *obsolete* wrt a state S at a time τ for any of the following reasons:

- The node is a goal, subgoal or precondition node and the node itself is achieved.
- The parent of the node is obsolete wrt S and τ. Indeed, if a node is obsolete there is no reason to plan for or execute any of its children (or descendants).

Thus, obsolete nodes amount to achieved goals, subgoals and preconditions and actions that have been introduced for them (and thus become redundant).

Definition 5. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, we define the set of* obsolete nodes *$Obsolete(S,\tau)$ as the set composed of each node $N \in Strategy \cup Goals$ of the form $N = \langle X, Pt\rangle$ such that:*

- *$Pt \in Obsolete(S,\tau)$ or*
- *$X = holds\_at(L,T)$ and $P_{plan} \cup KB_0 \models_{LP(\Re)} \Sigma \wedge holds\_at(L,T) \wedge T \leq \tau \wedge TC$*

A node is *timed out* wrt a state S at a time τ for any of the following reasons:

- It has not been achieved yet, and there is no way to achieve it in the future due to temporal constraints.
- Its parent or one of its siblings is timed out wrt S and τ. Indeed, if either the parent or a sibling of the node is timed out, there is no reason to keep the node for later planning. This condition is not imposed if the node is a top-level goal because top-level goals do not influence each other (expect via possible temporal constraints on their time variables).

Definition 6. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, we define the set of* timed out nodes *$TimedOut(S,\tau)$ as the set composed of each node $N \in Strategy \cup Goals$ of the form $\langle holds\_at(L,T), Pt\rangle$ such that:*

- *$N \notin Obsolete(S,\tau)$ and $\not\models_{\Re} \Sigma \wedge TC \wedge T > \tau$ or*
- *$Pt \in TimedOut(S,\tau)$ or*
- *$N \notin Goals$ and there exists $N' \in Siblings(N)$ such that $N' \in TimedOut(S,\tau)$).*

Using the above definitions we now define the revision transition which, roughly speaking, removes obsolete and timed out nodes.

Definition 7. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, the* revision transition *leads to a state $S' = \langle KB_0, \Sigma, Goals', Strategy', TC\rangle$ where, for each $N \in Strategy' \cup Goals'$:*

– $N \notin TimedOut(S, \tau)$, *and*
– *if* $N = \langle assume\_happens(O, T), \_ \rangle$ *then it is not the case that* $executed(O, \tau') \in KB_0$ *and* $T = \tau' \in \Sigma$, *and*
– *if* $N \in Obsolete(S, \tau)$ *then* $Parent(N) = \langle assume\_happens(O, T), \_ \rangle$, *and*
– $Parent(N) \in Goals' \cup Strategy'$.

Intuitively, each timed out node, each obsolete node and each executed action has to be eliminated from the tree. The only exception is represented by preconditions. Indeed, obsolete precondition at revision time are not eliminated because they must hold at execution time. If an obsolete precondition p for an action a is eliminated at revision time due to the fact that it holds at that time, something could happen later on (e.g. an external change or an action performed by some other agent or by the agent itself) that invalidates p so that it does not hold when a is executed. Note that we could also impose for the temporal constraints to be simplified at revision time, but this is not necessary to guarantee the correctness of our approach.

6 Selection functions

The planning and execution transitions require a *selection function* each. Here, we give possible definitions for these functions. Note that we use the term function loosely, as the selection randomly returns one of possibly several candidates.

6.1 Planning Selection Function

Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC \rangle$ at a time τ, the planning transition needs a *planning selection function* $SelP(S, \tau)$ to select a goal, subgoal or precondition G belonging to $Goals$ or $Strategy$, to be planned for. We define $SelP$ so that G satisfies the following properties:

– neither G nor any ancestor or sibling of G is timed out at τ;
– neither G nor an ancestor of G is achieved at τ; i.e. G is not obsolete and it does not hold at the current time;
– no plan for G belongs to S.

Definition 8. *Given a state* $S = \langle KB_0, \Sigma, Goals, Strategy, TC \rangle$ *at a time* τ, *the* planning selection function $SelP(S, \tau)$ *returns a goal, a subgoal or a precondition* $G = \langle holds\_at(L, T), \_ \rangle$ *such that:*

– $G \notin TimedOut(S, \tau)$;
– $G \notin Obsolete(S, \tau)$, *and it is not the case that* $P_{plan} \cup KB_0 \models_{LP(\Re)} holds\_at(L, T) \wedge T = \tau \wedge TC \wedge \Sigma$
– *there exists no* $G' \in Strategy$ *such that* $G = Parent(G')$;

Clearly it may be possible that a number of goals, subgoals and preconditions in a state satisfy the above properties and thus could be selected. We could further incorporate a number of heuristics to restrict the number of candidates G to be selected amongst.

6.2 Execution Selection Function

Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, the execution transition needs an *execution selection function* $SelE(S, \tau)$ to select an action A in $Strategy$ to be executed at τ. We define $SelE$ so that A satisfies the following properties:

- neither A nor any ancestor or sibling of A is timed out at τ;
- all preconditions (children) of A are satisfied at τ;
- no (goal, subgoal or precondition) ancestor of A is satisfied at τ;
- A has not been executed yet.

Definition 9. *Given a state $S = \langle KB_0, \Sigma, Goals, Strategy, TC\rangle$ at a time τ, the* execution selection function $SelE(S, \tau)$ *returns an action $A = \langle assume\_happens(O, T), \_\rangle$ such that:*

- $A \notin TimedOut(S, \tau)$*;*
- *for each $P = \langle holds\_at(P, T'), A\rangle \in Strategy$, $P \in Obsolete(S, \tau)$;*
- $A \notin Obsolete(S, \tau)$*;*
- *there exists no τ' such that $executed(O, \tau') \in KB_0$ and $T = \tau' \in \Sigma$.*

Again, heuristics could be incorporated within the execution selection function to restrict the number of selectable actions.

7 An example

In this section we show a simple example of life-cycle of an agent in the blocks-world domain of examples 1 and 2. We assume to have three blocks, a, b, c, all on the table initially. The formalisation of the initial configuration, using a special location $table$, is as follows:

$initially(on(a, table))$, $initially(on(b, table))$, $initially(on(c, table))$,
$initially(clear(a))$, $initially(clear(b))$, $initially(clear(c))$

Our objective is to have a tower with c on b on a by time 20. We can formalise this via top-level goals:

$G_1 = \langle holds\_at(on(b, a), T_1), \bot\rangle$ $G_2 = \langle holds\_at(on(c, b), T_2), \bot\rangle$

where $TC^0 = \{T_1 = T_2, T_1 \leq 20\}$

The following is a possible life-cycle of the agent, achieving G_1 and G_2.

Initial State: $S_0 = \langle\{\}, \{\}, \{G_1, G_2\}, \{\}, TC^0\rangle$
Time 1 - **Sensing Transition**: $\models^1_{Env}$ $\{\}$
Resulting state: $S_1 = S_0$
Time 2 - **Revision Transition**: There is nothing to be revised at this point.
Resulting state: $S_2 = S_1$
Time 3 - **Planning Transition**: Assume that $SelP(S_2, 3) = G_1$. Let (Δ, C) be the abductive answer wrt KB_{plan}, $\Delta_0 = \{assume\_holds(on(c, b), T_2)\}$ and $C_0 = TC^0$, where $\Delta = \{assume\_happens(mv(b, a), T_3)\}$ and $C = \{T_3 < T_1\}$. Let

$$
\begin{aligned}
Strategy^3 = \{\ & \langle assume\_happens(mv(b, a), T_3), G_1\rangle = A_1 \\
& \langle holds\_at(clear(a), T_4), A_1\rangle \\
& \langle holds\_at(clear(b), T_5), A_1\rangle\ \} \\
TC^3 = \ & TC^0 \cup C \cup \{T_4 = T_3, T_5 = T_3\}
\end{aligned}
$$

Resulting state: $S_3 = \langle\{\}, \{\}, \{G_1, G_2\}, Strategy^3, TC^3\rangle$

At this stage the tree structure is the one given earlier in the picture in Section 4.

Time 4 - **Execution Transition**: as the preconditions of action A_1 are both achieved at this time due to the $initially$ rules in KB_{plan}, then $A_1 = SelE(S_3, 4)$ (A_1 is the only action that can be selected at this time). Let

$KB_0^4 = \{executed(mv(b,a),3)$
$\Sigma^4 = \quad \{T_3 = 4\}$

Resulting state: $S_4 = \langle KB_0^4, \Sigma^4, \{G_1, G_2\}, Strategy^3, TC^3\rangle$

Time 5 - **Sensing Transition**: Assume that the sensing capability of the agent forces it to observe that b is actually on c at this time and that a is clear, namely $\models^5_{Env} \{on(b,c), \neg on(b,a), \neg on(c,table), \neg clear(c), clear(a)\}$. Basically, there has been either a problem in the execution of A_1 or an interference by some other agent. Then,

$KB_0^5 = KB_0^4 \cup \{\ observed(on(b,c),5), \quad observed(\neg on(b,a),5),$
$observed(\neg on(c,table),5),\ observed(\neg clear(c),5),$
$observed(clear(a),5)\}$

Resulting state: $S_5 = \langle KB_0^5, \Sigma^4, \{G_1, G_2\}, Strategy^3, TC^3\rangle$

Time 6 - **Revision Transition**: At this time the revision transition deletes from the strategy the action A_1 and its preconditions as A_1 has been executed.

Resulting state: $S_6 = \langle KB_0^5, \Sigma^4, \{G_1, G_2\}, \{\}, TC^3\rangle$

Time 7 - **Planning Transition**: Assume that the selected goal is again G_1, $SelP(S_6, 7) = G_1$. (Note that G_1 is again selectable as it is not achieved at time 7.) Similarly as for the previous planning transition, let:

$Strategy^7 = \{\ \langle assume\_happens(mv(b,a), T_3'), G_1\rangle = A_1'$
$\langle holds\_at(clear(a), T_4'), A_1'\rangle$
$\langle holds\_at(clear(b), T_5'), A_1'\rangle\ \}$
$TC^7 = \quad TC^3 \cup \{T_3' < T_1, T_4' = T_3', T_5' = T_3'\}$

Resulting state: $S_7 = \langle KB_0^5, \Sigma^4, \{G_1, G_2\}, Strategy^7, TC^7\rangle$

Time 8 - **Execution Transition**: as the preconditions of action A_1 are both achieved at this time, due to the $initially$ rules in KB_{plan} and to the observations in KB_0, then $A_1' = SelE(S_7, 8)$ (A_1' is the only action that can be selected at this time). Let

$KB_0^8 = \{executed(mv(b,a),8)$
$\Sigma^8 = \quad \{T_3' = 8\}$

Resulting state: $S_8 = \langle KB_0^8, \Sigma^8, \{G_1, G_2\}, Strategy^7, TC^7\rangle$

Time 9 - **Sensing Transition**: $\models^9_{Env} \{\}$

Resulting state: $S_9 = S_8$

Time 10 - **Revision Transition**: At this time the revision transition deletes from the strategy the action A_1' and its preconditions as A_1' has been executed.

Resulting state: $S_{10} = \langle KB_0^8, \Sigma^8, \{G_1, G_2\}, \{\}, TC^7\rangle$

Time 11 - **Planning Transition**: Assume that the selected goal is $SelP(S_{10}, 11) = G_2$. Note that at this time G_2 is the only goal that can be selected because goal G_1 is achieved. Similarly as for the previous planning transitions, let:

$Strategy^{11} = \{\ \langle assume\_happens(mv(c,b), T_6), G_2\rangle = A_2$
$\langle holds\_at(clear(a), T_7), A_2\rangle$
$\langle holds\_at(clear(b), T_8), A_2\rangle\ \}$
$TC^{11} = \quad TC^7 \cup \{T_6 < T_2, T_7 = T_6, T_8 = T_6\}$

Resulting state: $S_{12} = \langle KB_0^8, \Sigma^8, \{G_1, G_2\}, Strategy^{11}, TC^{11}\rangle$

Time 12 - **Execution Transition**: action A_2 is selected. Let
$KB_0^{12} = KB_0^8 \cup \{executed(mv(c,b), 12)$
$\Sigma^{12} = \quad \{T_3 = 4, T_3' = 8, T_6 = 12\}$
Resulting state: $S_{13} = \langle KB_0^{12}, \Sigma^{12}, \{G_1, G_2\}, Strategy^{11}, TC^{11}\rangle$
Time 13 - **Sensing Transition**: $\models_{Env}^{13}\ \{\}$
Resulting state: $S_{13} = S_{12}$
Time 14 - **Revision Transition**: At this time the revision transition deletes from the strategy the action A_2 and its preconditions as A_2 has been executed. Moreover as both G_1 and G_2 are achieved, the revision transition deletes them from the goals leading to a successful final state.
Resulting state: $S_{14} = \langle KB_0^{12}, \Sigma^{12}, \{\}, \{\}, TC^{11}\rangle$.

8 Related work and Conclusions

Planning has been a very active research and development area for some time. Systems have been developed for a range of applications such as medical, robotics and web services. Many approaches to planning have been proposed (e.g the STRIPS language with its improvements and related state-of-the-art systems such as Graphplan [1]). Here we concentrate on those closer to our work.
Our approach to planning is based on the abductive *event calculus*. It is thus closely related to Shanahan's abduction and event calculus planning work [14–18] and to the approach based on the *situation calculus*. The latter forms the basis of GOLOG [11], an imperative language implemented in PROLOG incorporating macro-actions (as procedures) and non-determinism. GOLOG has been shown to be suitable for implementing robot programs as high-level instructions in dynamic domains.
The contribution of our paper is in describing a system that allows partial planning and the interleaving of planning with sensing and executing actions. This integration is particularly suitable for (possibly resource bounded) agents situated in dynamic environments. Our partial plans, to some extent, have the flavour of the *compound actions* of Shanahan [16]. If well defined, both approaches allow us to find executable actions quickly. However, our formalisation is simpler than [16] as we do not need to use compound actions in our theories in order to achieve partial planning.
Compound actions are also exploited in the situation calculus, in particular [12] gives formal characterisations of compound actions and their preconditions and postconditions. Investigating how to incorporate them in our framework is subject of future work.
An important feature of our approach is the revision of the plans obtained by the Revision transition. The tree structure in the $Strategy$ part of each agent state allows an intelligent, selective way of revising the (partial) plan. This means that, if replanning becomes necessary, it is done only for unachieved goals and subgoals, thus avoiding the "replanning from scratch" method seen in [16].
There are issues that we have not addressed yet. These include ramification problems, which are addressed in [17] where it is pointed out that the *state-constraints* formalisation of ramifications can lead to inconsistencies. State-constraints are of the form

$holds\_at(P,T) \leftarrow holds\_at(P_1,T), \ldots, holds\_at(P_n,T)$

This rule can cause inconsistencies if, at a time t, P_1, ..., P_n and thus P hold. But at an earlier time, say t_1, $\neg P$ may hold and it is not clipped before the time t. As rules

of above form are needed to model subgoals, ramification is an important issue to be addressed. One way to avoid the problem of inconsistency could be to add, for each state constraint of the form above, another rule of the form

$declipped(P, T) \leftarrow holds\_at(P_1, T), \ldots, holds\_at(P_n, T)$

This approach is similar to the one that we have taken in the *bridge rules* of Section 3, but needs to be further investigated.

The *Sensing transition*, described in Section 5, is important for a situated agent, but is rather simplistic. It simply adds the observation to the agent's knowledge base and the *bridge rules* in the knowledge base perform some implicit conflict resolution. An alternative approach is presented in [16]. This proposal is that, once an observation is made, (possibly abductive) explanations of it are sought, thus avoiding some possible inconsistencies and giving a richer account of causes and effects. This approach has obvious disadvantages in cases where observations are such that the agent cannot be expected to find explanations for. E.g., in a communication scenario, an agent could observe that the network is down but has no way of knowing (or even guessing) why.

Another drawback of our Sensing transition is that it is random and passive. The agent collects information from the environment as a passive observer. An *active* form of sensing is described in [7, 2] where, as well as performing physical actions, the agent can perform active knowledge-producing (or sensing) actions. Such active sensing actions do not affect the external environment but they affect the agent's knowledge about the environment. Such an active sensing action can be performed, for example, to seek information from the environment about preconditions of actions before they are performed or to seek confirmation that an executed action has had its desired outcome. Active sensing actions are also addressed in [13] for imperative GOLOG programs where they allow conditional plans whose conditions are checked at "run-time".

An issue related to observations is that of *exogenous actions*. Our handling of observations combined with the Revision transition seem to be effective to capture both exogenous actions and their effects in the sense that, if our agent detects an action or a fact which invalidate a plan or a subplan already executed, the revision procedure will replan for that part (and only for that part). Another approach to exogenous (malicious) actions is that in [5] where, if exogenous actions change the external environment, a recovery procedure is performed with which the agent is able to restore the state to the one before the exogenous event occurred. With respect to our framework, drawbacks of that approach are that a number of assumptions have been made, in particular that the agent knows what kind of exogenous actions can be done and what their effects are. Also, this approach does not take into account the possibility that an exogenous action can "help" the agent to achieve its goals making certain subgoals and action unnecessary.

Finally, we remark that to properly evaluate our techniques, we are studying formal results such as soundness and completeness and we are doing practical experimentation with the CIFF system [4, 3] as the underlying abductive reasoner.

Acknowledgments

This work was partially funded by the IST programme of the EC, FET under the IST-2001-32530 SOCS project, within the Global Computing proactive initiative. The last author was also supported by the Italian MIUR programme "Rientro dei cervelli".

References

1. A. Blum and M. Furst. Fast planning through planning graph analysis. *AI*, 90:281–300, 1997.
2. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency for global computing: Computational model and prototype implementation. In *Proc. of Global Computing 2004 Workshop*. Springer Verlag, LNCS, 2004. To appear.
3. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. Abductive logic programming with CIFF: system description. In *Proc. JELIA04*, 2004. To appear.
4. U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. In *Proc. JELIA04*, 2004. To appear.
5. G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR'98*, pages 453–465, 1998.
6. J. Jaffar and M.J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19-20:503–582, 1994.
7. A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. ECAI2004*, 2004.
8. A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. OUP, 1998.
9. A. C. Kakas and R. Miller. A simple declarative language for describing narratives with ations. *Logic Programming*, 31, 1997.
10. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
11. H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of LP*, 31(1-3):59–83, 1997.
12. S. McIlraith and R. Fadel. Planning with complex actions. In *Proc. of NMR02*, 2002.
13. R. Scherl and H. J. Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144:1–39, 2003.
14. M. Shanahan. Event calculus planning revisited. In *Proceedings 4th European Conference on Planning*, pages 390–402. Springer Lecture Notes in Artificial Intelligence no. 1348, 1997.
15. M. Shanahan. *Solving the Frame Problem*. MIT Press, 1997.
16. M. Shanahan. Reinventing shakey. In *Working Notes of the 1998 AAAI Fall Symposium on Cognitive Robotics*, pages 125–135, 1998.
17. M. Shanahan. The ramification problem in the event calculus. In *Proc. of IJCAI99*, pages 140–146. Morgan Kaufmann Publishers, 1999.
18. M. Shanahan. Using reactive rules to guide a forward-chaining planner. In *Proceedings 4th European Conference on Planning*. Springer-Verlag, 2001.
19. K. Stathis, A. C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: a platform for programming software agents in computational logic. In J. Müller and P. Petta, editors, *Proc. AT2AI-4 – EMCSR'2004 Session M*, Vienna, Austria, April 13-16 2004.