Encoding Conformant Planning in A-Prolog

Michael Gelfond and A. Ricardo Morales

Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
{mgelfond,ricardo}@cs.ttu.edu

Abstract. This paper investigates the applicability of answer set programming for the design of provable correct and elaboration tolerant conformant planners. We describe an algorithm in which the search for a conformant plan is reduced to finding an answer set of a logic program, present some soundness and completeness results, and demonstrate that our solution is rather elaboration tolerant.

1 Introduction

Answer set programming (ASP) [MT99,B03] is a form of declarative programming in which:
(a) Knowledge relevant to a problem, $\mathcal{P}$, is represented by a program, $\mathcal{KB}$, in A-Prolog — a loosely defined collection of logic programming languages under the answer set (stable model) semantics [GL91].
(b) The task of solving $\mathcal{P}$ is reduced to finding the answer sets of $\mathcal{KB}$.
(c) Answer sets are computed by efficient answer set solvers [SNS02,DLVK02,LM04].

The goal of this paper is to investigate the applicability of ASP for the design of provably correct, and elaboration tolerant conformant planners. There is a substantial amount of work on the application of ASP to planning, but with the notable exception of [DLVK03], this work is limited to deterministic domains with complete initial situations. We are interested in removing these restrictions. In the first part of the paper we concentrate on the algorithm and some correctness results for conformant planning in deterministic domains with incomplete initial situations. In the second part we illustrate that our approach is rather elaboration tolerant — a simple modification of the problem (including the introduction of non-deterministic actions), leads to a simple modification of the problem's solution.

2 Specifying the Problem

A planning problem normally consists of a description of the planning domain, the agent's goal, and a (possibly incomplete) specification of its current state. We consider domains which can be represented by a transition diagram whose nodes are possible states of the domain and whose arcs are actions that take the domain from one state to another. Paths of the diagram correspond to possible trajectories of the system. We start by limiting our attention to transition diagrams which can be defined by action descriptions written in the action language $\mathcal{AL}$ from [BG00]. The signature Σ of an action description of $\mathcal{AL}$ consists of two disjoint, non-empty sets of symbols: the set $\mathbf{F}$ of fluents, and the set $\mathbf{A}$ of *elementary actions*. A set $\{e_1, \dots, e_n\}$ of elementary actions is called a *compound* action and it is interpreted as a collection of elementary actions performed simultaneously. By *actions* we mean both elementary and compound actions. By *fluent literals* we mean fluents and their negations. A set S of fluent literals is called *complete* if, for any $f \in \mathbf{F}$, $f \in S$ or $\neg f \in S$. An action description $\mathcal{D}$ of $\mathcal{AL}$ is a collection of statements of the form:

$$e \text{ causes } l \text{ if } p. \tag{1}$$

$$l \text{ if } p. \tag{2}$$

$$\text{impossible } a \text{ if } p. \tag{3}$$

where e is an elementary action, a is an action, l is a fluent literal, and p is a set of fluent literals from the signature Σ of $\mathcal{D}$. The set p is often referred to as the *precondition* of the corresponding statement. If p is empty the "if" part of the statement can be omitted. Statement (1), called a *dynamic causal law*, says that, if the elementary action e were to be executed in a state in which all literals of p hold, the fluent literal l will be caused to hold in any resulting state. (The restriction on e being elementary is not essential and can be lifted. We require it to simplify the presentation). Statement (2), often referred to as *state constraint*, says that any state satisfying p must satisfy l; Statement (3) is an *executability condition*. It says that action a cannot be performed in any situation in which p holds. Notice that here a can be compound, e.g. impossible $\{e_1, e_2\}$ means that elementary actions e_1 and e_2 cannot be performed concurrently.
In defining the transition diagram, $T(\mathcal{D})$, specified by an action description $\mathcal{D}$ we will follow [McT97].
A *state* σ of $T(\mathcal{D})$ is a complete, consistent set of literals closed under the state constraints of $\mathcal{D}$, i.e. for any state constraint (2) of $\mathcal{D}$, if $p \subseteq \sigma$ then $l \in \sigma$. An action a is said to be *prohibited* in a state σ if $\mathcal{D}$ contains an impossibility condition (3) such that p is true in σ. A transition, $\langle \sigma_1, a, \sigma_2 \rangle$, is defined as follows: Let $E(a, \sigma)$ stand for the set of all fluent literals l for which there is a causal law (1) in $\mathcal{D}$ such that $p \subseteq \sigma$. Given a set of literals S, $Cn(S)$ denotes the smallest set of fluent literals containing S closed under the state constraints of $\mathcal{D}$.

Definition 1. *[McT97]*
A transition $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$ iff a is not prohibited in σ_1 and

$$\sigma_2 = Cn(E(a, \sigma_1) \cup (\sigma_1 \cap \sigma_2)) \tag{4}$$

An action a is *executable* in state σ_1 if there is a state σ_2 such that $\langle \sigma_1, a, \sigma_2 \rangle \in T(\mathcal{D})$. A path $M = \langle \sigma_0, a_0, \sigma_1, \dots, a_{n-1}, \sigma_n \rangle$ of $T(\mathcal{D})$ is called a model of a chain

of events $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$. State σ_0 will be referred to as the initial state of M. We say M *entails* a set of fluent literals s, $M \models s$, if $s \subseteq \sigma_n$.
An action description $\mathcal{D}$ is called *deterministic* if for any state σ_1 and action a there is at most one successor state σ_2. Note that if $\mathcal{D}$ is deterministic there can be at most one model of α with initial state σ_0. In this case, we denote this model by $\langle \sigma_0, \alpha, \sigma_n \rangle$, and write $\sigma_n = \alpha(\sigma_0)$.
By a *partial state* of $\mathcal{D}$ we mean a consistent collection of fluent literals. Partial states and states are denoted by (possibly indexed) letters s and σ respectively.

Definition 2. A *planning problem* is a tuple $\langle \mathcal{D}, s_0, s_f \rangle$ where s_0 and s_f are partial states of $\mathcal{D}$.

Partial states s_0 and s_f characterize possible initial situations and the goal respectively.
A state σ containing a partial state s is called a *completion* of s. By $comp(s)$ we denote the set of all completions of s. An action a is *safe* in s if it is executable in every completion of s. $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ is *safe* in s if a_0 is safe in s and $\langle a_1, \ldots, a_{n-1} \rangle$ is safe in every state σ' such that for all $\sigma \in comp(s)$, $\langle \sigma, a_0, \sigma' \rangle \in T(\mathcal{D})$.

Definition 3. A chain $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ of events is a *solution* of a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, s_f \rangle$ if α is safe in s_0, and for every model M of α with the initial state $\sigma_0 \in comp(s_0)$, $M \models s_f$.

We often refer to α as a plan for s_f. If s_0 is a state and action description $\mathcal{D}$ is deterministic then α is a "classical" plan, otherwise it is a *conformant plan*.
To illustrate these definitions let us consider the following classical example.

Example 1. (Bomb in the toilet)
There is a finite set of toilets and a finite set of packages. One of the packages contains a bomb. The bomb can be disarmed by dunking the package that contains it in a toilet. Dunking a package clogs the toilet. Flushing a toilet unclogs it. Packages can only be dunked in unclogged toilets, one package per toilet. The objective is to find a plan to disarm the bomb.

The domain can be modeled by the following action description $\mathcal{TD}_0$:

$dunk(P, E)$ causes $\neg armed(P)$.
$dunk(P, E)$ causes $clogged(E)$.
$flush(E)$ causes $\neg clogged(E)$.
impossible $dunk(P, E)$ if $clogged(E)$.
impossible $dunk(P, E), flush(E)$.
impossible $dunk(P_1, E), dunk(P_2, E)$.
impossible $dunk(P, E_1), dunk(P, E_2)$.

E and P are variables for toilets and packages respectively; P_1 and P_2 stand for different packages. Similarly for toilets. Note that the last three statements specify physical impossibilities of some concurrent actions.

Let n and m denote the number of packages and toilets respectively. The initial situation, $s_0(n, m)$, of the problem will be given by a collection of literals:

$pkg(1), \dots, pkg(n)$.
$toilet(1), \dots, toilet(m)$.

together with a (possibly empty) collection of literals of the form $\neg armed(P)$.

The goal contains $\{\neg armed(1), \dots, \neg armed(n)\}$. We denote the corresponding planning problem by $B(n, m)$.

Action descriptions of $\mathcal{AL}$ containing no state constraints belong to the language $\mathcal{A}$ [GL93] and define deterministic transition diagrams. This is not necessarily true for arbitrary action descriptions of $\mathcal{AL}$. In the coming sections we will use the following condition which guarantees that the effects of actions are uniquely determined.

Definition 4. Action descriptions of $\mathcal{AL}$ whose state constraints contain at most one fluent literal in the body will be called *SC-restricted*.

Proposition 1. *[BG04]*
SC-restricted action descriptions of $\mathcal{AL}$ are deterministic.

In the next section we outline an algorithm for solving planning problems of $\mathcal{AL}$ using the methodology of ASP. Some results will provide sufficient conditions for correctness of our approach.

3 Solving Planning Problems in A-Prolog

In accordance with the ASP methodology our next step is to encode the corresponding planning problem $\mathcal{P}$ by a logic program whose answer sets contain information about the solutions of $\mathcal{P}$. We denote such a program by $e(\mathcal{P})$, and call it the *cautious encoding* of $\mathcal{P}$. The signature of $e(\mathcal{P})$ includes terms corresponding to fluent literals and actions of $\mathcal{D}$, as well as non-negative integers from the interval $[0, length]$ used to represent time steps. Atoms of $e(\mathcal{D})$ are formed by the following (sorted) predicate symbols: $holds(f, t)$ meaning "fluent f holds at time t"; $o(e, t)$ meaning "elementary action e occurs at time t"; and an auxiliary predicate symbol $ab(l, t)$.[1] We will use the following notation: If a is a compound action then $o(a, t) = \{o(e, t) : e \in a\}$. If l is a fluent literal then $h(l, t)$ stands for $holds(f, t)$ if $l = f$ and for $\neg holds(f, t)$ if $l = \neg f$. If p is a set of fluent literals,$h(p, t) = \{h(l, t) : l \in p\}$ and similarly $ab(p, t) = \{ab(l, t) : l \in p\}$. Predicate $\neg h(\neg f, t)$ will be identified with $h(f, t)$. By $\overline{l}$ we mean $\neg f$ if $l = f$ and f if $l = \neg f$. Literals $\overline{l}$ and l are called contrary literals. Similarly as above, $not\ \neg h(p, t) = \{not\ \neg h(l, t) : l \in p\}$. Finally, L and T are variables for fluent

[1] Some adjustment to this syntax is needed if one wants to use some of the existing answer set solvers. For instance, since SMODELS does not allow $ab(\neg f, t)$ we may replace it by, say, $ab(neg(f), t)$.

literals and time steps from $[0, length]$ respectively. The definition of $e(\mathcal{P})$ will be given in several steps:

Step One: The *cautious encoding* $e(\mathcal{D})$ of the action description $\mathcal{D}$ of $\mathcal{P}$ consists of rules obtained by replacing:

- Every causal law (1) in $\mathcal{D}$ by *effects and cancellation* rules of the form:

$$h(l, T+1) \leftarrow o(e, T), h(p, T). \tag{5}$$

$$ab(\bar{l}, T) \leftarrow o(e, T), not\ \neg h(p, T). \tag{6}$$

- Every state constraint (2) in $\mathcal{D}$, by rules of the form:

$$h(l, T) \leftarrow h(p, T). \tag{7}$$

$$ab(\bar{l}, T) \leftarrow ab(\overline{p}, T). \tag{8}$$

- Every impossibility condition (3) in $\mathcal{D}$, by logic programming constraints of the form:

$$\leftarrow o(a, T), not\ \neg h(p, T). \tag{9}$$

  - The inertia axiom:

$$h(L, T+1) \leftarrow h(L, T), not\ ab(L, T). \tag{10}$$

Note that the cancellation rules of the program guarantee that the inertia axiom for a fluent l is stopped if there is a possibility that $\bar{l}$ may be caused to be true in the next state. (The program $e(\mathcal{D})$ can be viewed as a modification and generalization of the translation from language $\mathcal{A}$ into A-Prolog [GL93]).

Step Two: The encoding $e(s_0)$, of the initial state:

$$e(s_0) = \{h(l, 0). : l \in s_0\}$$

Step Three: The cautious encoding, $e(s_f)$, of the goal of $\mathcal{P}$ consists of the rules:

- For every $l \in s_f$, $may\_fail(T) \leftarrow not\ h(l, T).$
- $goal(T) \leftarrow not\ may\_fail(T)$
- $success \leftarrow goal(T).$
- $\leftarrow not\ success.$
- $o(E, T)\ or\ \neg o(E, T).$

The first rule says that the goal of $\mathcal{P}$ may fail at time step T if there is $l \in s_f$ such that the truth value of l at T is either unknown, or known to be false. The second rule says that the goal is satisfied at time T if all the fluent literals from the goal are *known* to be true at T. The next two rules state that failing to achieve the goal is not an option. The last rule specifies that any (executable) elementary action e may occur at each time step $T \in [0, length]$. Intuitively the last rule generates a chain of events α with $|\alpha| < length$ while the rest of the program checks if α is a solution of problem $\mathcal{P}$. (Of course this description is purely conceptual and has nothing to do with the actual computation of answer sets).

Definition 5. *(Cautious encoding)*

$$e(\mathcal{P}) = e(\mathcal{D}) \cup e(s_0) \cup e(s_f) \tag{11}$$

The following establishes the relationship between answer sets of $e(\mathcal{P})$ and the solutions of $\mathcal{P}$.

Theorem 1. *(Soundness)*
Let $\mathcal{P} = \langle \mathcal{D}, s_0, s_f \rangle$ be a planning problem and S be an answer set of its cautious encoding $e(\mathcal{P})$ with $length = n$. If $\mathcal{D}$ is an SC-restricted action description of $\mathcal{AL}$ then $\alpha = \langle a_0, \ldots, a_{n-1} \rangle$ such that for every $0 \leq i < n$, $a_i = \{e : o(e, i) \in S\}$ is a solution of $\mathcal{P}$.

It is possible to show that the reverse of Theorem 1 does not holds. This is not surprising since the complexity of conformant planning (even without non-deterministic actions) is Σ_2^P [BKT99]. However for some classes of planning problems the cautious encoding is complete. We give the following simple sufficient condition guaranteeing completeness

Theorem 2. *(Completeness)*
Let $\mathcal{P} = \langle \mathcal{D}, s_0, s_f \rangle$ be a planning problem such that:

1. $\mathcal{D}$ contains no state constraints;
2. dynamic causal laws of $\mathcal{D}$ have no preconditions. Then $e(\mathcal{P})$ is complete with respect to $\mathcal{P}$.

These results suggest a method for finding a solution of a planning problem $\mathcal{P}$. First make sure that the action description $\mathcal{D}$ of $\mathcal{P}$ is deterministic. Next find an upper bound u limiting the possible lengths of your plans. (Of course this works only if such u can be known in advance which is frequently, but not always, the case.) Finally, let E_i to be a cautious encoding of $\mathcal{P}$ with $length = i$ and search for a conformant plan by repeatedly calling an answer set solver with E_i as an input. If an answer set S is found, then, by Theorem 1, the chain of events $\alpha(S)$ extracted from S is a (shortest) solution of $\mathcal{P}$. If no solution exists for $i \leq u$ and $\mathcal{P}$ satisfies the conditions of Theorem 2 then no such plan exists.
Cautious encoding is frequently used in conjunction with a collection, C, of constraints and definitions of A-Prolog specifying additional requirements which may be imposed on solutions of $\mathcal{P}$. Suppose for instance that we are interested in finding a *sequential* solution α of $\mathcal{P}$, i.e., solution $\alpha = \langle e_0, \ldots, e_{length-1} \rangle$ where e's are elementary actions. This requirement can be specified by the program C_0 which consists of the rule

$$act(T) \leftarrow o(E, T).$$

and constraints:

$$\leftarrow not\ act(T)$$

$$\leftarrow o(E_1, T), o(E_2, T), E_1 \neq E_2$$

The rule is the definition of $act(T)$ which says that some action occurred at time step T.

We want to be able to use the program $e(\mathcal{P}, C) = e(\mathcal{P}) \cup C$ to find solutions of $\mathcal{P}$ satisfying C. To make this precise we will need the following definitions. A constraint

$$\leftarrow L_1, \dots, L_m$$

of C will be called *basic* if L's are atoms of the form $o(e,t)$, $h(l,t)$, or their negations. We will say that a model $M = \langle \sigma_0, a_0, \sigma_1, \dots, a_{n-1}, \sigma_n \rangle$ of a chain of events $\alpha = \langle a_0, \dots, a_{n-1} \rangle$ *satisfies* L (and write $M \models L$) if

1. $M \models o(e,t)$ if $e \in a_t$;
2. $M \models h(l,t)$ if $l \in \sigma_t$;
3. $M \models \neg L$ if $M \not\models L$

$M \models \{L_1, \dots, L_k\}$ if for every $1 \leq i \leq k$, $M \models L_i$. This definition can, in a rather natural way, be expanded to include the defined literals. Finally, M satisfies a constraint $c = \{\leftarrow L_1, \dots, L_k\}$ if there is $0 \leq i \leq k$ such that $M \not\models L_i$.

Definition 6. A solution, α, of a planning problem $\mathcal{P} = \langle \mathcal{D}, s_0, s_f \rangle$ satisfies a constraint c ($\alpha \models c$) if every model M of α with the initial state $\sigma \in comp(s_0)$ satisfies c; α satisfies a set C of constraints if $\alpha \models c$ for every $c \in C$.

Now we can generalize Theorem 1.

Theorem 3. Let $\mathcal{P} = \langle \mathcal{D}, s_0, s_f \rangle$ be a planning problem, C be a collection of constraints and definitions described above, and S be an answer set of $e(\mathcal{P}, C)$ with $length = n$. If $\mathcal{D}$ is an SC-restricted action description of $\mathcal{AL}$ then $\alpha = \langle a_0, \dots, a_{n-1} \rangle$ such that for every $0 \leq i < n$, $a_i = \{e : o(e,i) \in S\}$ is a solution of $\mathcal{P}$ satisfying C.

Similar generalization can be proven for Theorem 2. To illustrate the use of constraints let us go back to the action description $\mathcal{TD}_0$ and planning problem $B(n,m)$ from Example 1. $\mathcal{TD}_0$ contains no state constraint and hence is SC-restricted. It is easy to check that by computing answer sets of $e(B(n,m))$ for $length = 0, 1, \dots$ we can find a conformant plan for disarming the bomb. This plan will be shortest but not necessarily the best; It can contain some useless concurrent actions like flushing unclogged toilets or getting rid of unarmed packages. Even though this can be dealt with by using specialized system directive (e.g. "minimize" of SMODELS) we prefer to describe the desired quality of plans by the following program C_1:

$$\leftarrow o(dunk(P,E),T), \neg h(armed(P),T).$$

$$\leftarrow o(flush(E),T), \neg h(clogged(E),T).$$

$$\leftarrow o(flush(E1),T), \neg h(clogged(E2),T), E1! = E2.$$

The planner is told not to dunk safe packages, to use unclogged toilets if possible, and not to flush unnecessarily. In addition to improving the quality of plans the constraints narrow the "search space" and hence make the planning process more efficient. It is not difficult to show that, if $B(n,m)$ has a solution then it has a solution satisfying the above constraints. Since the problem satisfies the conditions of Theorem 2 this means that if no solution of length u is found by our algorithm then such solution does not exist.

4 Adding New Knowledge

In this section we use Example 1 to show how new information about the domain can be incorporated in our planner. Due to the space limitations the discussion from now on will be substantially less formal.

Example 2. (State Constraints)
Suppose that a peculiar plumbing design guarantees that if toilet 1 is clogged then so is toilet 2. This can be recorded by the statement

$$clogged(2) \text{ if } clogged(1).$$

The resulting action description, $\mathcal{TD}_1$, does not belong to language $\mathcal{A}$. However, it is SC-restricted and hence, by Theorem 1, α returned by our algorithm for an input $\langle s_0, \mathcal{TD}_1, s_f \rangle$ is a solution of this problem.

The next example demonstrates how A-Prolog can be used to specify a fairly complex initial situation.

Example 3. (Complex initial situation)
Assume that the packages received by the character from our story are coming from people belonging to one of two hierarchically structured organizations, called b (bad) and g (good). The hierarchies are described in the usual way using relation $link(D_1, D_2)$ which indicates that a department D_1 is a subdivision of a department D_2. Organization g for instance can be represented by a collection $\{link(d_1, g).\ \ link(d_2, g).\ \ link(d_3, d_1).\ \ link(d_4, d_1).\}$
of atoms. It is known that packages sent by people from g are safe. The bomb is sent by someone working for b. There are packages labeled by the name of the department the sender works for, which can be recorded by the atom $from(P, D)$ - package P came from department D. There are also some unlabeled packages. The initial situation of the new problem, $B_h(n, m)$ will be described by the program H consisting of the above atoms and the following rules which define the organization the package came from and our trust in the good guys from g.

$$from(P, D_2) \leftarrow from(P, D_1), link(D_1, D_2).$$

$$\neg armed(P) \leftarrow from(P, g).$$

Naturally P ranges over packages and D's range over the departments. Our definition of a planning problem can be easily modified by allowing initial situation to be a program with a unique answer set. The corresponding algorithm and the correctness results can be easily adopted to this case. The initial situation of the new problem, $B_h(n, m)$ can be encoded by $e(s_0) = H \cup \{\neg h(armed(P), 0) \leftarrow \neg armed(P)\}$. (In general, the latter rule must be included for any fluent of the language of H). The rest of the cautious encoding, $e(B_h(n, m))$ will be unchanged. As before the problem $B_h(n, m)$ can be solved by computing answer sets of $e(B_h(n, m))$.

The next example shows how it is also possible to deal with non-deterministic actions. We extend $\mathcal{AL}$ by statements of the form

$$e \text{ causes } \{l_1, \ldots, l_n\} \text{ if } p. \tag{12}$$

which state that the execution of an elementary action e in a state satisfying p non-deterministically causes one of the fluent literals $l_1, \ldots, l_n$ to become true. The semantics of the new version of $\mathcal{AL}$ is a simple extension of the original semantics. The cautious encoding of (12) will consist of rules of the form

$$ab(\overline{l_i}, T) \leftarrow o(a, T), not\ \neg h(p, T). \tag{13}$$

for each $1 \leq i \leq n$.

Example 4. (Non-deterministic actions)
Let us consider a modification $\mathcal{TD}_2$ of TD_0 in which dunking a package may or may not clog the toilet. This can be encoded by a non-deterministic causal law

$$dunk(P, E) \text{ causes } \{clogged(E), \neg clogged(E)\}.$$

Other statements of TD_0 remain unchanged. The new planning problem will be denoted by $B_{nd}(n, m)$. It is not difficult to show that the new cautious encoding is sound and hence the models of $e(B_{nd}(n, m))$ determine conformant plans of $B_{nd}(n, m)$.

5 Conclusion and Future Work

In this paper we presented an ASP approach for representing planning problems with incomplete information about the initial situation and non-deterministic actions which is substantially different from the approach in [DLVK03], and showed some soundness and correctness results. We hope that the paper shows that the application of ASP to conformant planning is promising and deserves further investigation. The conditions for soundness and completeness presented in this paper are simple and sufficient, but limit the generality of the results. We are currently working on more general soundness and completeness results that will widen the scope of planning domains in which our approach is applicable and, for example, allow us to show completeness for the solution to Example 2. A comparison of the efficiency of our planning algorithm, called TTU, to other conformant planning systems is available in [GM03]. The TTU planner seems to be reasonably efficient in searching for comparatively short, highly parallel plans, in complex domains. We believe that new answer set solving algorithms and systems will substantially expand its applicability.

Finally we would like to mention the paper [SPB04] which uses ideas similar to ours. The relationship between both works is currently under investigation. We also would like to thank C. Castellini and P. Bertoli for their help with the $\mathcal{C}$-PLAN and HSCP planners.

References

[BKT99] Baral, C., Kreinovich V., and Trejo R. Computational complexity of planning and approximate planning in presence of incompleteness. In *Proc. of IJCAI-1999*. pp. 948-953.

[B03] Baral, C. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.

[BG04] Baral, C., and Gelfond, M. Logic Programming and Reasoning about Actions To appear in the Handbook of Temporal Reasoning in Artificial Intelligence.

[BG00] Baral, C., and Gelfond, M. Reasoning agents in dynamic domains. In Minker, J,. ed., *Logic-Based Artificial Intelligence*, Kluwer Academic Publishers, (2000), 257–279.

[BRC01] Bertoli P.G., Roveri M., and Cimatti A. Heuristic Search + Symbolic Model Checking = Efficient Conformant Planning In *Proc. of IJCAI-2001*. pp 467-472.

[CGT03] Castellini C., Giunchiglia, E., and Tacchella A. SAT-Based planning in complex domains: Concurrency, Constraints and Nondeterminism. In *Artificial Intelligence*, 147(1-2): 85-117 (2003).

[DLVK02] Leone N., Pfeifer G., Faber W., Calimeri F., Dell'Armi T., Eiter., Gottlob G, Ianni G., Ielpa G., Koch C., Perri S., and Polleres A. The DLV System. In *JELIA-8*, number 2424 in Lecture Notes in Computer Science, pp. 537-540, 2002.

[DLVK03] Eiter, T., Faber, W, Leone, N., Pfeifer, G., and Polleres, A. A logic Programming Approach to Knowledge State Planning: the $DLV^{\mathcal{K}}$ system. *Artificial Intelligence* 144, 1–2(Mar.) 157–211.

[GL91] Gelfond, M., and Lifschitz, V. Classical negation in logic programs and disjunctive databases. In *New Generation Computing*, 365–387, 1991.

[GL93] Gelfond, M., and Lifschitz, V. Representing Actions and Change by Logic Programs. *Journal of Logic Programming*, vol. 17, Num. 2,3,4, pp. 301–323, 1993.

[GM03] Gelfond, M., and Morales, A. R. On Conformant Planning in A-Prolog. *unpublished*, http://www.kralab.cs.ttu.edu/Papers 2003.

[LM04] Lierler Yu., and Maratea M., Cmodels-2: SAT-based Answer Sets Solver Enhanced to Non-tight Programs, In *Proc. of LPNMR-7*, pp. 346, 2004.

[MT99] Marek, W., and Truszczyński, M. Stable models and an alternative logic paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398, Springer Verlag, 1999.

[McT97] McCain, N., and Turner, H. Causal theories of action and change. In *Proc. of AAAI-97*, 460–465, 1997.

[SNS02] Simons P., Niemelä I., Soininen T. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2): 181-234 (2002)

[SPB04] Tran Cao Son, Phan Huy Tu, and Baral, C. Planning with Sensing Actions and Incomplete Information using Logic Programming. In *LPNMR'04* pp 261-274, Springer, 2004.