1 Introduction

One of the main problems in Knowledge Representation is the description of properties of actions and their effects [1, 13, 14, 15]. One of the most successful formalisms proposed in the last few years is the language $\mathcal{A}$, introduced by Gelfond and Lifschitz [7]. $\mathcal{A}$ is a simple "high-level" language, composed of two kinds of statements: statements about the effects of actions, and statements about states. We will describe the language in details in the next section. For now, as a "hello world" example of this language, we give the formalization of the well-known Yale Shooting Problem [9]:

$$
\begin{array}{l}
\textsf{initially } \neg Loaded \\
\textsf{initially } Alive \\
Load \textsf{ causes } Loaded \\
Shoot \textsf{ causes } \neg Alive \textsf{ if } Loaded \\
Shoot \textsf{ causes } \neg Loaded
\end{array}
$$

The first two statements mean that initially Fred is alive, and the gun is not loaded. The three other statements describe how the actions of loading and shooting the gun change the state of the world: for example Fred dies if the loaded gun is shot at him.

Many extensions of this language have been proposed in the literature, to incorporate indirect effects of actions [11], concurrent actions [2, 12], nondeterministic actions [4], dependences between fluents [8], hypothetical situations [3], etc.

In this paper we analyze the complexity of the basic language $\mathcal{A}$. Namely, given a set of statements, we show the complexity of deciding if it is consistent. We also prove the complexity of the entailment problem, that is, deciding what is implied by a set of statements.

In the general case (when any kind of propositions is allowed) the complexity turns out to be NP complete and coNP complete, respectively for the consistency and entailment problems. Thus, in the general case these problems are intractable. This is quite a surprising result, since the language does not involve any logical connective (such as or, and, etc), and the semantics is not based on a minimization of any kind.

Given the fact that the general language is intractable, one may ask for restrictions of the language with tractable consistency checking and entailment. Besides the benefit of knowing when the problem is tractable and when it is not, this analysis allows us to determine what the source of intractability of $\mathcal{A}$ is. Namely, we will show that the basic source of hardness is the incompleteness of knowledge of the initial state. However, it is not the only source of intractability: we show that some limitations on the effects of the actions lead to tractability. We will give a longer discussion at the end of the paper, after that all the technical results have been formally stated and proved.

The paper is organized as follows: in the next section we recall the definition of $\mathcal{A}$ and the basic concepts of computational complexity. In Section 3 we show the complexity of the complete (unrestricted) language. In section 4 we reformulate the problem of consistency in $\mathcal{A}$ as satisfiability of the completion of a set of clauses. In Section 5 we show which restrictions of $\mathcal{A}$ lead to tractability and which do not. The proof of NP hardness of the unrestricted language is given in details, while the proofs under restrictions are sketched. The effect of states that are unreachable from the initial state is outlined in Section 6. Finally, in Section 7 we discuss the implications of the results of this work.

2 Definitions

A detailed description of the language $\mathcal{A}$ is given in [7], where more examples are also provided. For the sake of completeness, we recall here the syntax and the semantics of $\mathcal{A}$. More details, including the translation of $\mathcal{A}$ into logic programs, can be found in the above paper.

2.1 Syntax of $\mathcal{A}$

There are two nonempty sets of symbols, called fluent names and action names. A fluent expression is a fluent name possibly preceded by the negation symbol $\neg$. If F is a fluent expression and $A_1, \ldots, A_m$ are action names, then

$$F \text{ } \mathsf{after} \text{ } A_1; \ldots; A_m$$

is a value proposition. If $m = 0$, the above statement is written

$$\mathsf{initially} \text{ } F$$

A value proposition specifies the truth value of a fluent in the initial state, or after a sequence of actions. An effect proposition is an expression of the form

$$A \text{ } \mathsf{causes} \text{ } F \text{ } \mathsf{if} \text{ } P_1, \ldots, P_m$$

where A is an action name and $F, P_1, \ldots, P_m$ are fluent expressions. The meaning is of course that the action A, when performed, makes the fluent expression F true, if the fluents $P_1, \ldots, P_m$ are true. The fluents $P_1, \ldots, P_m$ are called preconditions, while F is the effect or postcondition of the proposition. If $m = 0$ the if-clause can be dropped, obtaining the shorter version

$$A \text{ } \mathsf{causes} \text{ } F$$

A domain description is a set of value and effect propositions.

2.2 Semantics of $\mathcal{A}$

The semantics of $\mathcal{A}$ is defined in terms of *states* and *models*. A state is a set of fluent names. A fluent expression F is true in the state σ if $F \in \sigma$, false otherwise. A fluent expression $\neg F$ is true in σ if and only if F is false in σ. A transition function Φ is a function from the set of pairs (A, σ), where A is an action and σ a state, to the set of states. With $\Phi(A, \sigma)$ we want to represent the state obtained performing the action A in the state σ.

Let $V_D^+(A, \sigma)$ be the set of the fluent names F (i.e. positive fluent expressions) such that there exists an effect proposition A causes F if $P_1, \ldots, P_m$ in the domain description D and $P_1, \ldots, P_m$ are true in σ. Intuitively, $V_D^+(A, \sigma)$ represents the set of fluents whose value must become true when the action A is performed in the state σ.

In a similar manner, $V_D^-(A, \sigma)$ is the set of fluents whose value must became false, and thus is defined as the set of fluent names F such that there exists an effect proposition A causes $\neg F$ if $P_1, \ldots, P_m$ in the domain description D and $P_1, \ldots, P_m$ are true in σ.

The transition function associated to a domain description D is the (partial) function Ψ_D defined as

$$\Psi_D(A,\sigma) = \begin{cases} (\sigma \cup V_D^+(A,\sigma)) \backslash V_D^-(A,\sigma) & \text{if } V_D^+(A,\sigma) \cap V_D^-(A,\sigma) = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

An interpreted structure is a pair (σ_0, Φ), where σ_0 is a state (the initial state of the system) and Φ is a transition function. Given an interpreted structure $M = (\sigma_0, \Phi)$ and a sequence of actions $A_1; \ldots; A_m$, we denote the state after the sequence as

$$\Phi(A_1; \ldots; A_m, \sigma_0) = \Phi(A_m, \Phi(A_{m-1}, \ldots, \Phi(A_1, \sigma_0) \ldots))$$

A value proposition F after $A_1; \ldots; A_m$ is true in the structure M if F is true in the state $\Phi(A_1; \ldots; A_m, \sigma_0)$, false otherwise.

An interpreted structure $M = (\sigma_0, \Phi)$ is a model of the domain D if and only if the transition function Φ is equal to Ψ_D, and is total (i.e. defined for each pair (A, σ)) and each value proposition in the domain is true in M. Different models of the same domain description differ only by their initial state.

A domain description is consistent if and only if there is a model of it. A domain description D entails a value proposition V (written $D \models V$) if and only if V is true in any model of D.

2.3 Computational Complexity

In this section we recall the basic concepts and definitions of computational complexity, namely the definition of the classes NP, coNP and the concept of hardness and completeness.

We assume that the reader is familiar with the basic concepts of computational complexity. We use the standard notation of complexity classes that can be found in [10]. Namely, the class P denotes the set of problems whose solution can be found in polynomial time by a *deterministic* Turing machine, while NP denotes the class of problems that can be resolved in polynomial time by a *non-deterministic* Turing machine. The class coNP denotes the set of decision problems whose complement is in NP. We call NP-hard a problem G if any instance of a generic NP problem can be reduced to an instance of G by means of a polynomial-time (many-one) transformation. A problem is NP complete if it is in NP and is also NP hard. The same for coNP hardness and completeness.

Clearly, P $\subseteq$ NP and P $\subseteq$ coNP. We assume, following the mainstream of computational complexity, that these containments are strict, that is P $\neq$ NP and P $\neq$ coNP. Therefore, we call a problem that is in P *tractable*, and a problem that is NP-hard or coNP-hard *intractable* (in the sense that any algorithm resolving it would require a super-polynomial amount of time in the worst case).

The prototypical NP-complete problem is deciding the truth of the expression $\exists X.\Pi$, where Π is a set of clauses, each composed of three literals. This expression is true if and only if there exists a truth assignment to the letters of X that satisfies Π. A coNP-complete problem is deciding whether a set of clauses (each composed of three literals) is not satisfiable, that is, if there is no truth assignment that satisfies it.

3 Basic Results

In this section we show the complexity of the problems of consistency and entailment for the language $\mathcal{A}$ without restrictions. We first prove that if the initial state is given, then consistency is polynomial. Using this result, we are able to prove that consistency is in general NP complete.

Lemma 1 *Given D, deciding whether Ψ_D is total is polynomial.*

Proof. The function Ψ_D is total if and only if for each action A and fluent F, if there are two effect propositions A causes F if $P_1, \ldots, P_m$ and A causes $\neg F$ if $R_1, \ldots, R_k$, then there exists a fluent G such that $G \in \{P_1, \ldots, P_m\}$ and $\neg G \in \{R_1, \ldots, R_k\}$ (or vice-versa). If this condition is verified, then for any state σ the sets $V_D^+(A, \sigma)$ and $V_D^-(A, \sigma)$ are disjoint. On the other side, if this condition is not verified for a pair of effect propositions, then, given the state

$$\sigma = \{F_i \text{ fluent name} \mid P_i = F_i\} \cup \{F_j \text{ fluent name} \mid R_i = F_j\}$$

we have that $V_D^+(A, \sigma)$ and $V_D^-(A, \sigma)$ are not disjoint. □

Now we can prove that, given the initial state, checking consistency is polynomial.

Theorem 1 *Given a domain description D and an initial state σ_0, deciding if $M = (\sigma_0, \Psi_D)$ is a model of D, is polynomial.*

Proof. To prove that M is a model of D, we have to prove that Ψ_D is total, and that each value proposition in D is true in M. The first is polynomial by the previous lemma.

In order to prove that $M = (\sigma_0, \Psi_D)$ is a model of D, we notice that all the following steps are polynomial:

1. Given A, D and σ, determining $V_D^+(A, \sigma)$ and $V_D^-(A, \sigma)$ is polynomial.
2. As a result, the state $\Psi_D(A, \sigma)$ can be computed in polynomial time, for any A, D and σ.
3. Finally, to verify whether a value proposition F after $A_1; \ldots; A_m$ is true in M, determine the state

$$\Psi_D(A_m, \Psi_D(A_{m-1}, \ldots \Psi_D(A_1, \sigma_0) \ldots))$$

in polynomial time, and then check whether F is true in this state.

For each value proposition in the domain D, check in polynomial time whether it is true in M. The whole procedure is thus polynomial. □

In general, the initial state is not given. We must check if there exists an initial state σ_0 such that each value proposition of D is true in $M = (\sigma_0, \Psi_D)$. This is the kind of problems that are in NP.

Theorem 2 *Deciding if a domain description D is consistent is* NP *complete.*

Proof. The membership of the problem in NP follows from the above theorem: to prove that D is consistent, guess an initial state σ_0 and then in polynomial time determine whether $M = (\sigma_0, \Psi_D)$ is a model of D.

In order to prove hardness, we give a reduction from the problem 3sat to it. Given a set of clauses $\Pi = \{\gamma_1, \ldots, \gamma_m\}$, each composed at most of three literals over the alphabet X, we give the following domain description: the set of fluents is $X \cup \{F\}$ (that is, there is a fluent for each variable of Π, and an additional fluent F), there is only one action A, and the propositions in the domain description are:

$$\begin{array}{l} \textsf{initially } F \\ A \textsf{ causes } \neg F \textsf{ if } \neg L_1, \neg L_2, \neg L_3 \\ \qquad \text{for each clause } \gamma_i = L_1 \vee L_2 \vee L_3 \text{ in } \Pi \\ F \textsf{ after } A \end{array}$$

That is, for each clause γ_i we invert its three literals, and then we build an effect proposition with them as preconditions. This domain description D has a model if and only if the given set of clauses is satisfiable.

Informally, the proof goes as follows. Since F is true in the initial state, and there is no other initially proposition regarding the value of the x_i's in the initial state, the initial states σ_0 are indeed in one-to-one correspondence with the interpretations over the alphabet of Π. If all the fluents in the triple $\{L_1, L_2, L_3\}$ corresponding to a clause γ_i are false in the initial state σ_0, then the fluent F is false after A. On the other side, γ_i (and thus Π) is false w.r.t. an interpretation if and only if all its three literals are false. As a result, F is true after A if and only if the set of clauses is true. Since the third proposition forces F to be true after the action A, the given domain description is consistent if and only if the set of clauses is consistent.

Now we formally prove that the given reduction works. Let us suppose Π satisfiable, and let I be a model of Π. This implies that each $\gamma_i \in \Pi$ is satisfied by I. Let $\sigma_0 = I \cup \{F\}$. We prove that (σ_0, Ψ_D) is a model of D. First of all, Ψ_D is total. Second, we prove that $F \in \Psi_D(A, \sigma_0)$, which implies that F after A is true in the interpreted structure. Since $\gamma_i = L_1 \vee L_2 \vee L_3$ is satisfied by I, at least a fluent expression in $\{L_1, L_2, L_3\}$ is true in σ_0. As a result, at least one fluent expression in $\{\neg L_1, \neg L_2, \neg L_3\}$ is false, thus $V_D^-(A, \sigma_0) = \emptyset$. Since all the effect propositions in D have a negative effect, $V_D^+(A, \sigma_0) = \emptyset$ as well. As a result, $\Psi_D(A, \sigma_0) = \sigma_0$, that contains F.

We prove the converse, that is, if the domain description is consistent then Π is satisfiable. Let (σ_0, Ψ_D) be a model of D. We prove that $\sigma_0 \backslash \{F\}$ is a model of Π. Since (σ_0, Ψ_D) is a model of D, and D contains A after F, we have $F \in \Psi_D(A, \sigma_0)$. Consider a generic clause $\gamma_i = L_1 \vee L_2 \vee L_3$. If all the fluent expressions L_1, L_2, and L_3 were false in σ_0, then it would be $F \notin \Psi_D(A, \sigma_0)$. As a result, at least one fluent among L_1, L_2, and L_3 must be true in σ_0, thus γ_i is satisfied by the interpretation $\sigma_0 \backslash \{F\}$. Since this is true for each $\gamma_i \in \Pi$, the set Π is satisfied by the interpretation $\sigma_0 \backslash \{F\}$. □

Note that in this proof we used only *one* action, and each effect proposition has at most three preconditions. In the next section we will prove that one precondition in each proposition suffices to give the NP completeness of the consistency checking.

The second problem of interest in the language $\mathcal{A}$ is the entailment, that is, deciding if a fluent is true or false after a sequence of actions (given a domain description). This is closely related to the problem of consistency. Actually, to prove that D entails F after $A_1; \ldots; A_m$ suffices to prove that $D \cup \{\neg F$ after $A_1; \ldots; A_m\}$ is inconsistent[1].

[1]This rule holds only for the original semantics of $\mathcal{A}$, and not, for example, for the modified semantics of Section 6.

Theorem 3 *Deciding if a value proposition is entailed by a domain description is* coNP *complete.*

In this case, we derive complexity of entailment from the complexity of the consistency checking. However, this is not always possible: with some restrictions, for example, the consistency is always guaranteed (and thus the complexity of consistency is polynomial!) but the entailment is still coNP complete.

4 $\mathcal{A}$ as Completion

In this section we show a translation of $\mathcal{A}$ into Clark's completion [5]. Since the completion of a set of propositional clauses is a propositional formula, this is (indirectly) a translation from $\mathcal{A}$ into propositional logic.

This translation gives two main benefits. The first is computational. The language $\mathcal{A}$ has been developed only recently, while Clark's completion has been deeply analyzed, and the computational properties of propositional logic are well-known. Many heuristics has been developed for solving the problem of satisfiability of a set of propositional clauses: indeed, this is the most studied NP complete problem. In principle, any NP complete problem can be translated into propositional logic. The reduction presented here is quite simple, and "almost" modular (each pair effect proposition–value proposition is translated into a set of clauses). This allows for example the individuation of tractable subclasses of $\mathcal{A}$. Since the each proposition is translated into one or more clauses, it is possible to obtain a set of clauses in a specified form (e.g. Krom) by imposing constraints on the propositions. For example, if the effect propositions have no precondition, then the corresponding set of clauses can be simplified in order to obtain a set of clauses of two literals. Satisfiability of such sets of clauses is polynomial.

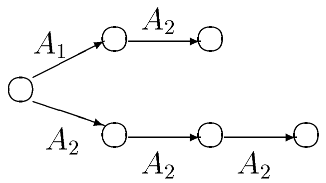
Another benefit is the possibility of using heuristics developed for propositional formulas. For instance, GSAT [18] and approximate entailment [17] are defined for propositional formulas. The translation presented here is simple and intuitive, but an empirical validation is needed. This is left as an open problem of this work.

Finally, during the public review of this paper, Erik Sandewall pointed out a similarity between the completion version of $\mathcal{A}$ and the entailment method PMON, defined in the framework of Features and Fluents [16]. A detailed explanation is contained in the discussion page of this paper http://www.ida.liu.se/ext/etai/received/actions/001/aipf.html.

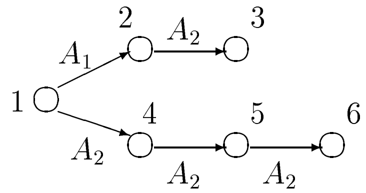
Let us start with some definitions. Given a domain description D, we define the history tree associated with it to be a tree with edges labeled with actions, and such that for each value proposition F after $A_1;\ldots;A_m$ there is a branch from the root to a node, such that its edges are labeled $A_1,\ldots,A_m$. For example, if D is

$$
\begin{aligned}
D \;=\{\; & \textsf{initially } F_1, \\
& \textsf{initially } F_2, \\
& F_1 \textsf{ after } A_1, \\
& \neg F_1 \textsf{ after } A_1;A_2, \\
& \neg F_2 \textsf{ after } A_2;A_2;A_2, \\
& A_1 \textsf{ causes } F_1 \textsf{ if } F_2, \\
& A_2 \textsf{ causes } \neg F_2 \;\}
\end{aligned}
$$

then the corresponding history tree is



Put a unique numeric label in each node, starting from 1 in the root



Let l be the number of nodes in the tree. Given the set of fluents $\{F_1, \ldots, F_n\}$, we define the alphabet of the corresponding boolean formula as $\{x_o^j \mid 0 \leq j \leq l,\ 1 \leq o \leq n\} \cup \{c_o^j \mid 1 \leq j \leq l,\ 1 \leq o \leq n\}$. The variable x_o^j is used to denote that the fluent F_o is true in the state corresponding to the node j. The variable c_o^j is used to denote that the truth value of the fluent F_o must change, in the state corresponding to the node j, that is, in the state j the fluent F_o must assume a value that is the opposite of the value it had in the previous state. Each proposition in the domain description is translated into a clause of a set K, as follows.

Initially Propositions. We start from the initially propositions. For each initially F_o we have a formula x_o^1 in K. For each initially $\neg F_o$ we have $\neg x_o^1$ in K. For each F_o that does not appear in any initially proposition, we have the pair of clauses $x_o^0 \leftarrow x_o^1$ and $x_o^1 \leftarrow x_o^0$. This meaningless pair of clauses is necessary to avoid the completion to set $\neg x_o^1$.

After propositions. For each F_o after $A_1; \ldots; A_m$ we add x_o^j in K, where j is the label of the node at the end of the path $A_1; \ldots; A_m$. For $\neg F_o$ after $A_1; \ldots; A_m$, we add $\neg x_o^j$.

Effect propositions. The translation of effect proposition is a little bit more complex. For each adjacent pair of nodes i and j, that are linked by an edge labeled A_k from i to j, we add the pair of clauses

$$\begin{array}{lcl} x_o^j & \leftarrow & c_o^j, \neg x_o^i \\ x_o^j & \leftarrow & \neg c_o^j, x_o^i \end{array}$$

for each o. The meaning is: the fluent F_o is true after the action A if and only if it was true in the previous state, and it must not change, or it was false, and it must change. Now, consider the effect propositions that contain A_k. Suppose that the effect is positive, that is, the effect proposition is (the symbol $[\neg]$ denote a possible negation)

$$A_k \ \mathsf{causes}\ F_o \ \mathsf{if}\ [\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$$

Then, add in K the clause

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, \neg x_o^i$$

If the effect is negative, translate

$$A_k \ \mathsf{causes}\ \neg F_o \ \mathsf{if}\ [\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$$

into

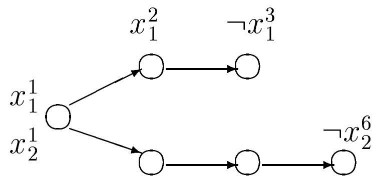
$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, x_o^i$$

This set of clauses is used to formalize the fact that the truth value of the fluent F_o must change, if the state before the action satisfies the necessary preconditions.

The result of this translation is a set of clauses K. We prove that the domain description is consistent if and only if the Clark's completion of K is consistent.

We first show the result of applying the algorithm above to the domain description D defined at the beginning of the section. We have 6 nodes, and 2 fluents. Thus, $(6 + 1) \times 2 \times 2$ variables are required. As we will see, not all of them are really required.

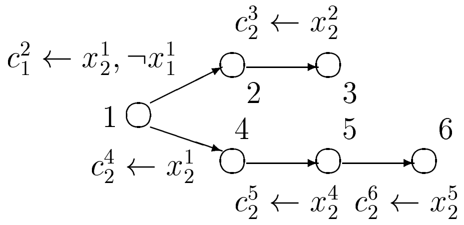
The first phase is to translate the value propositions into rules on variables. Namely, we have



Then, for each edge (i, j), we add the clauses

$$\begin{array}{lcl} x_o^j & \leftarrow & c_o^j, \neg x_o^i \\ x_o^j & \leftarrow & \neg c_o^j, x_o^i \end{array}$$

Finally, we add the effect propositions.



The resulting set of clauses is the union of all the clauses in these figures. In this case there are few clauses, since there are only two fluents and two events. This is just a toy example. The reason of why this set of clauses is not equivalent to the domain description D is that the truth of a variable x_o^j does not impose the truth of variables x_o^i, that is, there is no backward projection. This is achieved by the completion.

Theorem 4 *If Ψ_D is total, the domain description D is consistent if and only if the completion of K is consistent.*

Proof. By hypothesis, the function Ψ_D is total. As a result, we have to prove that

1. If there exists an initial state σ_0 such that all the value propositions in the domain description are true in (σ_0, Ψ_D), then the completion of the set of clauses K has a model.

2. If the completion of K has a model, then there exists an initial state σ_0 such that (σ_0, Ψ_D) satisfies all the value propositions in D.

We prove each statement separately.

Comp(K) **sat.** Since each x_o^j is in the head of at least one clause, the completion does not add clauses $\neg x_o^j$. On the converse, there are c_o^j's that do not appear in the head of clauses. However, they do not appear in the body of other clauses, either, thus we do not need to consider them any more.

Let I be the model of the completion of K. We prove that (σ_0, Ψ_D) is a model of D, where

$$\sigma_0 = \{F_o \mid x_o^1 \in I\}$$

This is equivalent to prove that each value proposition is true in the interpreted structure (σ_0, Ψ_D). We prove first an intermediate lemma: $x_o^j \in I$ if and only if $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$, where j is the label of the node at the end of the path $A_1; \ldots; A_m$. The proof is by induction on the length of the path.

Base Step: By definition, $x_o^1 \in I$ if and only if $F_o \in \sigma_0$.

Induction Step: The induction hypothesis is

$$x_o^i \in I \quad \Leftrightarrow \quad F_o \in \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$$

where i is the parent of j. The thesis to prove is

$$x_o^j \in I \quad \Leftrightarrow \quad F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$$

Let us assume $x_o^j \in I$. By hypothesis, K contains two clauses with x_o^j in the head, namely $x_o^j \leftarrow c_o^j, \neg x_o^i$ and $x_o^j \leftarrow \neg c_o^j, x_o^i$. Since $x_o^j \in I$, at least one conjunction between $c_o^j \wedge \neg x_o^i$ and $\neg c_o^j \wedge x_o^i$ must be true in the model I.

$c_o^j \wedge \neg x_o^i$ **true.** Since c_o^j is true in I, at least one clause with c_o^j in the head must have its body satisfied by I. Since $x_o^i \notin I$, this clause must be of the form

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, \neg x_o^i$$

thus, $[\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i$ are true in I. By hypothesis, $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ are true in $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$, and since D contains the effect proposition A_k causes F_o if $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ we have also $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

$\neg c_o^j \wedge x_o^i$ **true.** This implies $x_o^i \in I$, thus $F_o \in \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$. Moreover, $c_o^j \notin I$. From the definition of completion it follows that the bodies of all clauses with c_o^j in the head must be false. The clauses with $\neg x_o^i$ in the body are false because $x_o^i \in I$. For any other clause

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, x_o^i$$

there must be an index o_s such that $[\neg]x_{o_s}^i$ is false. As a result, $[\neg]F_{o_s}$ is false in the state $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$, thus for each effect proposition A_k causes $\neg F_o$ if $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$, at least one of its preconditions is false in the state $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$.

As a result, F_o is true in the previous state, and each effect proposition with $\neg F_o$ as effect has at least one false precondition. This implies $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

Let us assume, on the converse, that $x_o^j \notin I$. Since K contains the clauses $x_o^j \leftarrow c_o^j, \neg x_o^i$ and $x_o^j \leftarrow \neg c_o^j, x_o^i$, both the bodies of these clauses must be false. As a result, there are two possible cases: that $c_o^j \wedge x_o^i$ is true, and that $\neg c_o^j \wedge \neg x_o^i$ is true. For each of these two cases, we will prove that $F_o \notin \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

$c_o^j \wedge x_o^i$ **true.** Since c_o^j is true, it must be true at least one body of a clause that have c_o^j in the head. Since x_o^i is true in I, this clause must not be one with $\neg x_o^i$ in the body, thus this clause has the form

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, x_o^i$$

As a result, all the fluent expressions $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ are true in the state $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$. Since the effect proposition A_k causes $\neg F_o$ if $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ is in D, this implies $F_o \notin \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

$\neg c_o^j \wedge \neg x_o^i$ **true.** Since c_o^j is false, all the clauses with c_o^j in the head must have a body that is false in I. The clauses with x_o^i in the body already satisfies this property. On the converse, for all the clauses

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, \neg x_o^i$$

there must be a literal $[\neg]x_{o_s}^i$ that is false in I. As a result, $[\neg]F_{o_s}$ is false in the state $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$, thus no effect proposition may make F_o true in the state $\Psi_D(A_1; \ldots; A_m, \sigma_0)$. Since x_o^i is false in I, we have also $F_o \notin \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$, thus $F_o \notin \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

So far, we have proved that $x_o^j \in I$ if and only if $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$. Now, we have to prove that each value proposition of D is true in the interpreted structure (σ_0, Ψ_D). Let us assume F_o after $A_1; \ldots; A_m \in D$. The set of clauses K thus contains x_o^j, where j is the label of the node at the end of the path $A_1; \ldots; A_m$. As a result, $x_o^j \in I$, which implies $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$ by the previous lemma. By definition, F_o after $A_1; \ldots; A_m$ is true in (σ_0, Ψ_D).

D **consistent.** Let (σ_0, Ψ_D) be a model of D. Let

$$\begin{aligned} I \;=\; & \{x_o^j \mid F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)\} \cup \\ & \{c_o^j \mid F_o \in \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0) \backslash \Psi_D(A_1; \ldots; A_m, \sigma_0) \cup \\ & \quad \Psi_D(A_1; \ldots; A_m, \sigma_0) \backslash \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)\} \cup \\ & \{x_o^0 \mid F_o \in \sigma_o\} \end{aligned}$$

where, as usual, j is the label of the node at the end of the path from the root labeled $A_1; \ldots; A_m$. We prove that I is a model of the completion of K.

There are three kinds of clauses in K. The clauses corresponding to value propositions are unit clauses, thus they are not modified by the completion. These are trivially satisfied. The same for clauses $x_o^1 \leftarrow x_o^0$ and $x_o^0 \leftarrow x_o^1$, whose completion is $x_o^1 \equiv x_o^0$.

Now, consider the clauses with c_o^j in the head. We prove that the completion of all these clauses is satisfied by I. In order to prove this, we first assume c_o^j true in I, and prove that there is at least one clause in K whose body is true in I. Then, we assume c_o^j false and prove that all the clauses with c_o^j in the head have a body that is false in I.

c^j_o **true.** This means that

$$\begin{array}{lcl} F_o & \in & \Psi_D(A_1;\ldots;A_{m-1},\sigma_0) \\ F_o & \notin & \Psi_D(A_1;\ldots;A_m,\sigma_0) \end{array}$$

or vice versa. We consider the other case afterwards.

Let i be the parent node of j, and A_k be the label of the edge (i,j). The first equation implies that $x^i_o \in I$. The second one implies that there exists an effect proposition

$$A_k \ \textsf{causes} \ \neg F_o \ \textsf{if} \ [\neg]F_{o_1},\ldots,[\neg]F_{o_k}$$

such that the fluent expressions $[\neg]F_{o_1},\ldots,[\neg]F_{o_k}$ are true in the state $\Psi_D(A_1;\ldots;A_{m-1},\sigma_0)$. As a result, $[\neg]x^i_{o_1},\ldots,[\neg]x^i_{o_k}$ are true in I.

Since $F_o \in \Psi_D(A_1;\ldots;A_{m-1},\sigma_0)$, we have also $x^i_o \in I$. The set K contains the clause

$$c^j_o \leftarrow [\neg]x^i_{o_1},\ldots,[\neg]x^i_{o_m},x^i_o$$

whose body is true.

Now, consider the case

$$\begin{array}{lcl} F_o & \notin & \Psi_D(A_1;\ldots;A_{m-1},\sigma_0) \\ F_o & \in & \Psi_D(A_1;\ldots;A_m,\sigma_0) \end{array}$$

Let i be the parent node of j, and A_k be the label of the edge (i,j). The first equation implies that $x^i_o \notin I$. The second one implies that there exists an effect proposition

$$A_k \ \textsf{causes} \ F_o \ \textsf{if} \ [\neg]F_{o_1},\ldots,[\neg]F_{o_k}$$

such that the fluent expressions $[\neg]F_{o_1},\ldots,[\neg]F_{o_k}$ are true in the state $\Psi_D(A_1;\ldots;A_{m-1},\sigma_0)$. As a result, $[\neg]x^i_{o_1},\ldots,[\neg]x^i_{o_k}$ are true in I.

Since $F_o \notin \Psi_D(A_1;\ldots;A_{m-1},\sigma_0)$, we have also $x^i_o \notin I$. The set K contains the clause

$$c^j_o \leftarrow [\neg]x^i_{o_1},\ldots,[\neg]x^i_{o_k},\neg x^i_o$$

whose body is true.

c^j_o **false.** We have to prove that all the clauses with c^j_o is in the head have a body that is false in I. There are two possible cases. The first is

$$\begin{array}{lcl} F_o & \in & \Psi_D(A_1;\ldots;A_{m-1},\sigma_0) \\ F_o & \in & \Psi_D(A_1;\ldots;A_m,\sigma_0) \end{array}$$

the second one is with F_o that does not belong to any of the above two states. Consider the former case.

The first equation implies $x^i_o \in I$, thus the clauses with $\neg x^i_o$ in the body have a body which is false in I. Consider the generic clause with c^j_o in the head and x^i_o in the body

$$c^j_o \leftarrow [\neg]x^i_{o_1},\ldots,[\neg]x^i_{o_m},x^i_o$$

Since this clause is in K, the domain description D must contain an effect proposition

$$A_k \textsf{ causes } \neg F_o \textsf{ if } [\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$$

but, since $F_o \in \Psi_D(A_1; \ldots; A_m, \sigma_0)$, at least one fluent expression in $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ must be false. As a result, the body of each clause with c_o^j in the head is false.

Now, consider the case

$$\begin{array}{rcl} F_o & \notin & \Psi_D(A_1; \ldots; A_{m-1}, \sigma_0) \\ F_o & \notin & \Psi_D(A_1; \ldots; A_m, \sigma_0) \end{array}$$

This is the converse of the previous case. We prove that the body of each clause with c_o^j in the head is false. The first equation implies $x_o^i \notin I$, thus the bodies that contain x_o^i are false. Consider the body of a clause that contains $\neg x_o^i$:

$$c_o^j \leftarrow [\neg]x_{o_1}^i, \ldots, [\neg]x_{o_m}^i, \neg x_o^i$$

Since this clause is in K, the domain description D must contain an effect proposition

$$A_k \textsf{ causes } F_o \textsf{ if } [\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$$

Since $F_o \notin \Psi_D(A_1; \ldots; A_m, \sigma_0)$, at least one fluent expression in $[\neg]F_{o_1}, \ldots, [\neg]F_{o_m}$ must be false. As a result, the body of each clause with c_o^j in the head is false.

In order to complete the proof, we must prove that the completion of the clauses with x_o^j in the head are satisfied by I. The completion of such clauses is $x_o^j \equiv (c_o^j \not\equiv x_o^i)$. This is equivalent to $c_o^j \equiv (x_o^j \not\equiv x_o^i)$. By hypothesis, c_o^j is true in I if and only if F_o is true in the state $\Psi_D(A_1; \ldots; A_{m-1}, \sigma_0)$ but not in $\Psi_D(A_1; \ldots; A_m, \sigma_0)$, or vice versa. As a result, c_o^j is true in I if and only if x_o^i is true and x_o^j is false, or vice versa. Thus, $c_o^j \equiv (x_o^j \not\equiv x_o^i)$ is true in I.

This completes the proof. □

Note that the totality of Ψ_D can be checked in polynomial time.

From a purely theoretical point of view, this translation is polynomial. Moreover, the completion of a set of clauses is a propositional formula whose size is linear in the size of the initial set.

However, the aim of this translation is that heuristics for sat can be applied to the resulting propositional formula. Since the size of the formula and the number of its variables are crucial for the efficiency of these algorithms, we should reduce it as much as possible. It makes perfectly sense to spend more time on reducing the size of the formula, even if this phase is not particularly efficient, since the next step is the solution of an NP complete problem.

The size of K, and thus that of the completion of K, can be reduced. For example if A_k has never F_o as effect, then for each edge (i,j) labeled A_k, there is no clause with c_o^j in the head. Thus, the completion of K has $\neg c_o^j$. Thus the completion of the pair of clauses $x_o^j \leftarrow c_o^j, \neg x_o^i$ and $x_o^j \leftarrow \neg c_o^j, x_o^i$ is equivalent to the completion of $x_o^j \leftarrow x_o^i$. This clause can also be deleted from K by replacing each occurrence of x_o^j with x_o^i.

Another way of reducing the size of K is by making explicit the knowledge already in D. For example, if there is an proposition $\mathsf{initially}\ F$, and there is an edge from the root labeled A_k, then the effect propositions like $A_k\ \mathsf{causes}\ \ldots\ \mathsf{if}\ \ldots, \neg F, \ldots$ can be neglected.

We consider now an algorithm for producing a shorter set of clauses K that corresponds to a domain description D, in the sense that the completion of K will be consistent if and only if D is consistent.

The set of variables of K is $X = \{x_i\}$. The number of variables needed will be determined by the algorithm. The algorithm is based on a visit of the tree. We use a vector V with n elements, where n is the number of fluents. Each element can be true, false, or a variable of X.

We start on the root, and initialize the vector. The i-th element of V is true if $\mathsf{initially}\ F_i \in D$, is false if $\mathsf{initially}\ \neg F_i \in D$, otherwise is x_i. The vector V is essentially a three-valued interpretation over the alphabet of fluents. When the i-th element is a variable, we mean that the value of the fluent F_i is not determined.

We perform a visit of the history tree. Suppose we are on an edge (l, m), labeled A_k. We construct a set of effect propositions E. Copy all the effect propositions $A_k\ \mathsf{causes}\ F_i\ \mathsf{if}\ F_{i_1}, \ldots, F_{i_r}$ in E. If there is an F_{i_j} such that the i_j-th element of V is false, delete the proposition from E. If there is an element F_{i_j} such that the i_j-th element of V is true, delete F_{i_j} from the proposition. If, after these operations, there is an effect proposition $A_k\ \mathsf{causes}\ F_i$ without preconditions, then the i-th element of V becomes true. The same for $A_k\ \mathsf{causes}\ \neg F_i$.

The result of this step is a "minimal" set of effect propositions E, each having A_k. Consider the fluent F_i. If there is no effect proposition with F_i or $\neg F_i$ as effect, do nothing. If all the effect propositions in E have F_i positive as effect, then for each $A_k\ \mathsf{causes}\ F_i\ \mathsf{if}\ F_{i_1}, \ldots, F_{i_r}$ in E, put

$$x_s \leftarrow v_{i_1}, \ldots, v_{i_r}$$

and $x_s \leftarrow v_i$ in K, where x_s is a new variable of the alphabet not already used, and v_i is the i-th element of V. The new i-th element of V is x_s.

If there are effect propositions with F_i as effect, and some other with $\neg F_i$, then convert each $A_k\ \mathsf{causes}\ F_i\ \mathsf{if}\ F_{i_1}, \ldots, F_{i_r}$ into

$$x_c \leftarrow v_{i_1}, \ldots, v_{i_r}, \neg v_i$$

and convert $A_k\ \mathsf{causes}\ \neg F_i\ \mathsf{if}\ F_{i_1}, \ldots, F_{i_r}$ into

$$x_c \leftarrow v_{i_1}, \ldots, v_{i_r}, v_i$$

Add also

$$\begin{array}{l} x_s \leftarrow x_c, \neg v_i \\ x_s \leftarrow \neg x_c, v_i \end{array}$$

Now, consider the value propositions. If D contains a value proposition $F_i\ \mathsf{after}\ A_1; \ldots; A_r$, such that $A_1; \ldots; A_r$ is the sequence of labels of the path from the root to m, then the i-th element of V becomes true. If it was false, then D is inconsistent. If it was an x_i, add $x_i = \mathsf{true}$ in K. The same for propositions $\neg F_i\ \mathsf{after}\ A_1; \ldots; A_r$.

This step must be repeated for all the paths in the history tree. We can prove that the resulting K has a consistent completion if and only if D is consistent.

Theorem 5 *If Ψ_D is total, the domain description D is consistent if and only if the completion of the set of clauses K built by the above algorithm is consistent.*

The new set of clauses is shorter than the original one. Let us consider for example the number of propositional variables. If v is the size of the value propositions (i.e. each value proposition F after $A_1; \ldots; A_k$ has size k) and f is the number of fluents of the domain description, then the set of clauses has $O(v \cdot f)$. This is the worst case: for each action in each value proposition there are $2 \cdot f$ propositional variables.

The modified set of clauses has a number of variables that depends on the possible effect of actions. Let $e(A_j)$ be the number of fluent modified by the action A_j, that is, $e(A_j) = |\{F_i \; . \; A_j \text{ causes } [\neg]F_i \in D\}|$. The reduced set of clauses have $O(v \cdot \max_{A_j} e(A_j))$ variables. This translation reduces the number of variables if there are actions that do not change the value of some fluents.

5 Restrictions

In this section we show some restrictions of $\mathcal{A}$ that make the problems of consistency checking and entailment tractable, and try to determine under which conditions the problems are tractable and when they are not, by proving that some other restrictions of the language are still NP complete.

5.1 Intractable Restrictions

First of all, we give the general method used to prove intractability of $\mathcal{A}$ under restrictions. All the proofs of NP hardness are by reduction from the NP complete problem 3sat: given a set of clauses of three literals, decide if it is satisfiable. To prove the hardness of $\mathcal{A}$ we have to show that there exists a polynomial function that, given a set of such clauses, gives a domain description that is consistent if and only if the set of clauses is satisfiable.

We informally explain how the description is obtained from the set of clauses. First of all, we define the domain description D corresponding to Π in such a way that Ψ_D is total. Given Π, we have one fluent name for each atom of the alphabet of Π. The initially propositions do not contain any of these fluents. We have also at least another fluent F, which is specified to be true in the initial state (i.e., initially $F \in D$).

The domain description defined so far has exactly one possible initial state σ_0 for each interpretation I over the alphabet of Π. The idea is to include in the domain description some effect propositions in such a way that, starting from the initial state σ_0, the fluent F becomes false after a given sequence of actions $A_1; \ldots; A_m$ if and only if Π is false in the interpretation I.

The last proposition of the domain description is the value proposition F after $A_1; \ldots; A_m$. This way, D has a model if and only if the set Π is satisfiable. The proof of Theorem 2 is an example of this technique: the action A makes F false if one of the clauses of Π is false (as to say, if Π as a whole is false). Since there is a proposition A after F, the domain D has a model if and only if it is possible to satisfy the set of clauses.

5.1.1 Constant number of actions.

As shown in the Section 3, even if there is only *one* action in the domain, the problem of consistency checking is still NP complete. And this holds even if there is a limit on the number of preconditions (three) of the effect propositions. Namely, Theorem 2 can be rewritten as

Theorem 6 *The problem of consistency checking in $\mathcal{A}$ is* NP *complete, even if there is only one action, and each effect proposition has at most three preconditions.*

5.1.2 One precondition.

As already noted, the NP completeness may seem surprising, since the language contains only literals, and boolean connectives are not allowed (there are more expressive dialects of $\mathcal{A}$ in which they are present). Essentially, the $\wedge$ is "simulated" by effect propositions like A causes F if F_1, F_2, which causes F to be true if both F_1 and F_2 are true, while the $\vee$ is obtained by A causes F if F_1 and A causes F if F_2, which cause F to be true if either F_1 or F_2 is true. However, since $\{\vee, \neg\}$ are enough to express the propositional calculus, effect propositions with two or more preconditions are not necessary.

Theorem 7 *Verifying the consistency of a domain description is* NP *complete even if there are only two actions, and each effect proposition has at most one precondition.*

Proof. The following domain description is consistent if and only if the set of clauses $\Pi = \{\gamma_1, \ldots, \gamma_m\}$ is consistent:

$$
\begin{array}{l}
\textsf{initially } F \\
\left.\begin{array}{l}
\textsf{initially } \neg F_j \\
A_1 \textsf{ causes } F_j \textsf{ if } L_1 \\
A_1 \textsf{ causes } F_j \textsf{ if } L_2 \\
A_1 \textsf{ causes } F_j \textsf{ if } L_3 \\
A_2 \textsf{ causes } \neg F \textsf{ if } \neg F_j
\end{array}\right\} \text{for each } \gamma_j = L_1 \vee L_2 \vee L_3 \text{ in } \Pi \\
F \textsf{ after } A_1; A_2
\end{array}
$$

In words: the three effect propositions of A_1 causes F_j to be true if and only if at least one literal of γ_j is true in the initial state. The effect proposition of A_2 causes F to become false if at least one F_j is false in the previous state, that is, if and only if there is a false clause. As a result, if Π is inconsistent, there is no initial state that satisfy D, since at least one clause will be false, and thus at least an F_j is false. On the converse, if Π is satisfiable, then there is at least an initial state σ_0 such that all the F_j are true in the state $\Psi_D(A_1, \sigma_0)$, and thus the value proposition F after $A_1; A_2$ is satisfied. □

5.1.3 Limited effects of action.

Another (perhaps more natural) restriction to impose is that an action can have only a limited number of effects, that is, the number of effect propositions in which a given action appear is bounded by a constant.

This would be useful, since often an action influences only a small part of the world of interest. However, even if each action modifies one fluent, the problems are still intractable.

Theorem 8 *Consistency checking is* NP *complete even if each action appears only in one effect proposition, and each effect proposition has at most one precondition.*

Proof. The reduction is similar to that of Theorem 2. Since we cannot use A causes $\neg F$ if $\neg L_1, \neg L_2, \neg L_3$, we replace A with a sequence of actions $A_1, \ldots, A_m$ in the following manner

$$A_j \textsf{ causes } \neg F \textsf{ if } \neg L_1, \neg L_2, \neg L_3$$

As a result, the value proposition F after $A_1, \ldots, A_m$ is true if and only if all the clauses of Π are satisfied. Of course, we can simulate the above effect proposition with three other effect propositions, as in the theorem above, obtaining a domain description with only one precondition in each effect proposition. □

5.1.4 Positive fluents.

The last intractable class we consider is the one in which we assume that in the effect propositions the fluents appear only positive, that is, no symbol $\neg$ is allowed in effect propositions. This is equivalent to say that only positive fact may cause (positive) effects. For example, in the YSP, the propositions *Shoot* causes $\neg$*Alive* if *Loaded* and *Shoot* causes $\neg$*Loaded* do not respect this restriction, while the other ones do.

Theorem 9 *Consistency checking is* NP *complete even if preconditions and effects are always positive, and each effect proposition has at most two preconditions.*

Proof. The easiest way is to invert the sign of all the fluents of the proof of Theorem 2. Thus, we have the domain description

$$\begin{array}{l} \textsf{initially } \neg F \\ A \textsf{ causes } F \textsf{ if } L_1, L_2, L_3 \\ \neg F \textsf{ after } A \end{array}$$

which is consistent if and only if the domain description of Theorem 2 is consistent, thus iff Π is satisfiable. We can also replace A causes F if L_1, L_2, L_3 with two effect propositions

$$\begin{array}{l} A_1 \textsf{ causes } F \textsf{ if } L_1, L_2 \\ A_2 \textsf{ causes } F \textsf{ if } F, L_3 \end{array}$$

Of course, these are not positive propositions, since each L_i can be positive or negative. In order to overcome this problem, we use two fluents for each atom in Π, one for each literal. In order to prove the theorem, we impose that, in the initial state, the fluent corresponding to a negative literal is false if and only if the fluent corresponding to the positive one is true. This is done by the propositions

$$\begin{array}{lll} \textsf{initially } \neg G & & \\ A \textsf{ causes } G \textsf{ if } P_1 & A \textsf{ causes } G \textsf{ if } P_2 & G \textsf{ after } A \\ A \textsf{ causes } L \textsf{ if } P_1, P_2 & \neg L \textsf{ after } A & \end{array}$$

in each model, if any, of these propositions, P_1 must be the opposite of P_2, that is, P_1 is true in the initial state if and only if P_2 is false, and vice-versa. □

5.2 Tractable restrictions

5.2.1 No after.

Assume that the only value propositions allowed are of the form initially F. In this case, given that no pair initially F and initially $\neg F$ is in the domain, and Ψ_D is total, the description is always consistent, since no proposition F after A can be in contradiction with the evolution of the system. Since Ψ_D is total, contradiction cannot arise from propositions like A causes G and A causes $\neg G$: in such cases, Ψ_D is not total (the totality of Ψ_D can be checked in polynomial time). Formally, given an initial state σ_0, the interpreted structure (σ_0, Ψ_D) is a model of the domain if and only if Ψ_D is total (which is polynomial to verify) and each value proposition in D is true in the model. Since the only value propositions allowed are initially F, this amounts to verify that F is true in σ_0. As a result, D is consistent if and only if the transition function Ψ_D is total and there are no pair of value propositions initially F and initially $\neg F$. Both these properties can be checked in polynomial time.

In this case, the entailment is still coNP complete. This happens because D entails F after $A_1; \ldots; A_m$ if and only if $D \cup \{\neg F$ after $A_1, \ldots, A_m\}$ is inconsistent. This domain description does contain a value proposition with actions, and thus it does not respect the restriction. Indeed, in the proof of Theorem 2 one value proposition suffices to prove hardness (in the reduction used, the initially proposition can be removed).

5.2.2 Effect propositions without preconditions.

In this case, consistency and entailment are easy to verify, since the evolution of the system can be determined regardless of the initial state.

5.2.3 Totally specified initial state.

When, for each fluent name F there is in D a value proposition initially F or initially $\neg F$, the initial state is fully specified. By Theorem 1, the consistency is polynomial, since there is at most one model.

5.2.4 One positive precondition and positive effects.

When each effect proposition has at most one positive precondition, and the effect is also positive, consistency can be verified in polynomial time. Informally, this is a form of monotonicity, since fluents can be added to the initial state, but not deleted. It can be proved that in this case the consistency checking is polynomial.

Theorem 10 *When all the effect propositions in the domain description have only one positive precondition and a positive effect, then the check of consistency is polynomial.*

Proof. Consider the history tree of the domain description D. We build a formula that is consistent if and only if D is consistent, and the check of satisfiability on this formula is polynomial.

Let F_o after $A_1; \ldots; A_m$ be an value proposition in the domain, let j be the node in the history tree at the end of the path $A_1; \ldots; A_m$, and i be its parent. Start with x_o^j. Consider all the effect propositions

$$A_m \text{ causes } F_o \text{ if } F_{o_s}$$

Replace x_o^j with $x_o^i \vee x_{o_1}^i \vee \cdots \vee x_{o_k}^i$ Repeat the above procedure for each variable in this clause, until we obtain a disjunction of atoms in $\{x_s^1\}$. The result is a positive clause.

For value propositions whose fluent is negative, the result of the same procedure is the negation of a positive clause, that is, a conjunction of negative atoms. The domain description D is consistent if and only if the set of these formulas is consistent. This check can be done in polynomial time. □

5.2.5 Small classes of fluents

Suppose that the set of fluent names $\mathcal{F}$ can be partitioned in small disjoint sets $\mathcal{F}_1, \ldots, \mathcal{F}_r$, that is, $\bigcup \mathcal{F}_i = \mathcal{F}$, and $\mathcal{F}_i \cap \mathcal{F}_j = \emptyset$ and there exists a constant k such that $|\mathcal{F}_i| \leq k$ for each $\mathcal{F}_i$. If, for each effect proposition A causes F if $P_1, \ldots, P_m$ in the domain there exists a $\mathcal{F}_i$ such that $F, P_1, \ldots, P_m \in \mathcal{F}_i$, then the problem of consistency is polynomial.

6 Unreachable States

In this section we consider domain descriptions in which some states are not reachable from the initial state. We start showing an example in which an intuitively correct description does not have a model. The world of interest is a simple base 3 counter, that is, a counter that starts from 0, and when an action is executed, becomes 1, then 2, then 0 again and so on.

We formalize this domain with two fluents H and L representing the high and low bit of the counter, and one action I (increase the counter). The initial state is $\emptyset$ (both bits false), and when the action I is executed the first time it becomes $\{L\}$ (that represents 01), the second time $\{H\}$ (that is 10) and then $\emptyset$ again.

One description of this domain is the following.

$$
\begin{array}{l}
\mathsf{initially}\ \neg H \\
\mathsf{initially}\ \neg L \\
I\ \mathsf{causes}\ L\ \mathsf{if}\ \neg H, \neg L \\
I\ \mathsf{causes}\ H\ \mathsf{if}\ L \\
I\ \mathsf{causes}\ \neg L\ \mathsf{if}\ L \\
I\ \mathsf{causes}\ \neg H\ \mathsf{if}\ H
\end{array}
$$

one can see that there is only one possible initial state, namely $\emptyset$ (both bits false) and the action I has the following effects.

$$
\begin{array}{l}
\sigma_0 = \emptyset \\
\Psi_D(I, \sigma_0) = \{L\} \\
\Psi_D(I, \Psi_D(I, \sigma_0)) = \{H\} \\
\Psi_D(I, \Psi_D(I, \Psi_D(I, \sigma_0))) = \emptyset \\
\vdots
\end{array}
$$

This corresponds exactly to behavior of the counter. However, the above domain description does not have a model. To see that, note that $V_D^+(I, \{H, L\})$ $= \{H\}$ and $V_D^-(I, \{H, L\}) = \{H, L\}$, thus the function Ψ_D is not determined for the state $\{H, L\}$ and the action I, that is, there is no possible outcome performing the action I in the state $\{H, L\}$.

Note that the state $\{H, L\}$ is never a state of the system: starting from $\sigma_0 = \emptyset$ and applying any sequence of actions, the system never arrives in the state $\{H, L\}$. Thus, the above domain description is intuitively correct, but it has no model under the classical semantics of $\mathcal{A}$.

To take into account such unreachable states, we consider a slightly different definition of model. First of all, we say that a value proposition F after $A_1; \ldots; A_m$ is weakly true in the model $M = (\sigma_0, \Phi)$ if and only if F is true in the state $\Phi(A_1; \ldots; A_m, \sigma_0)$, or there exists a $k < m$ such that $\Psi_D(A_1; \ldots; A_k, \sigma_0)$ is undefined. A state is said to be a possible initial state for a domain description D if and only if each value proposition of D is weakly true in (σ_0, Ψ_D).

A state σ is reachable from σ_0 if and only if there exists a sequence of actions $A_1; \ldots; A_m$ such that $\sigma = \Psi_D(A_1; \ldots; A_m, \sigma_0)$.

Finally, an interpreted structure (σ_0, Φ) is a model of the domain description D if and only if σ_0 is a possible initial state of D and Φ coincides with Ψ_D, whenever defined, and is defined for all the actions A and all the states reachable from any possible initial state. The domain description D entails (under the semantics of reachable states) a value proposition V (written $D \models_R V$) if and only if V is true in all the models of D.

In the example of the counter, $\sigma_0 = \emptyset$ is the only possible initial state. The states reachable from it are $\{L\}$ and $\{H\}$. For domain descriptions without unreachable states, the new semantics coincides with the classical semantics of $\mathcal{A}$.

It is now possible to show that any (possibly inconsistent) domain description D with unreachable states can be modified in order to obtain a new one D' that is consistent. This modified description entails (under the classical semantics of $\mathcal{A}$) exactly the same value propositions that D entails under the semantics of reachable states. In the above example of the counter, the last proposition of the domain can be replaced with I causes $\neg H$ if $H, \neg L$.

In general, this can be done in the following way: for each pair of effect propositions

$$A \text{ causes } F \text{ if } P_1, \ldots, P_m$$
$$A \text{ causes } \neg F \text{ if } R_1, \ldots, R_m$$

such that there is no fluent name G such that $G \in \{P_1, \ldots, P_m\}$ and $\neg G \in \{R_1, \ldots, R_m\}$ or vice versa, replace the first proposition with

$$A \text{ causes } F \text{ if } P_1, \ldots, P_m, \neg R_1$$
$$\vdots$$
$$A \text{ causes } F \text{ if } P_1, \ldots, P_m, \neg R_m$$

Repeat this step, if there is another pair of effect propositions with the above condition verified.

At the end, we obtain a new domain description D' that is consistent under the classical semantics and has the same entailed formulas of D. About the correctness of the procedure we give the following theorem.

Theorem 11 *If D is consistent under the semantics of reachable states, and D' is the domain description obtained from D applying the above algorithm, then for any value proposition V it holds*

$$D \models_R V \quad \textit{iff} \quad D' \models V$$

In general, the domain description D' obtained from D using the above algorithm has size exponential w.r.t. the size of D. This can be avoided, for example allowing boolean formulas as preconditions of the effect propositions.

About the complexity of the new semantics, we note that the determination of unreachable states might be in general more difficult than the problem of finding models. We prove that the problem of reachability is in PSPACE. We believe that it is also PSPACE complete.

Theorem 12 *Given a set of effect propositions D, and two states σ_1 and σ_2, deciding if σ_2 is reachable from σ_1 is in* PSPACE.

Proof. The following polynomial-space algorithm determines whether a state σ_2 is reachable from σ_1 in less than k steps, in the domain description D.

```
boolean Reach(σ1, σ2, k)
if σ1 = σ2 then return true
if k = 1 then
  if ∃A such that Ψ_D(A, σ1) = σ2
   then return true
   else return false
foreach σ3 do
  if Reach(σ1, σ3, k/2) ∧ Reach(σ3, σ2, k/2)
   then return true
return false
end
```

This algorithm runs using only a polynomial amount of space since the two calls of *Reach* can be done in sequence. Then, at each time only a logarithmic number of procedure calls are active. The result of $Reach(\sigma_1, \sigma_2, 2^n)$, where n is the number of fluent names, is the reachability of σ_2 from σ_1. □

Under the new semantics, the consistency checking of D can be made off-line. This means that once decided the consistency of D, the problem of entailment is only in coNP. Thus, one can determine the consistency for the domain description D once, and then the complexity of the entailment is only coNP. This is useful in all the cases in which a domain description must be queried w.r.t. several queries.

Finally, we remark that this semantics is different from the previous ones, notably [2]. Indeed, in that paper the authors introduce a variant of $\mathcal{A}$ that allows the treatment of concurrent actions. In their semantics, the domain description $\{A \textsf{ causes } F, A \textsf{ causes } \neg F\}$ is consistent (in the dialect of $\mathcal{A}$ developed by Baral and Gelfond, a domain description without value propositions is always consistent). This is reasonable in the context analyzed there, but it is not in the context of deterministic, non-concurrent actions, where the above domain description should be inconsistent.

7 Conclusions

In this paper we have investigated the computational complexity of the language $\mathcal{A}$ for reasoning about actions. The main problem analyzed here is the consistency of a domain description expressed in this language. The complexity of this problem turns out to be NP complete, for the general (unrestricted) case. The reason of the intractability is basically that the initial state is in general not fully specified.

Some tractable subclasses of the language have been discussed. We tried also to establish a boundary between tractability and intractability. The most significant tractable class is the set of domain descriptions with a fully specified initial state. Another important one is the class of descriptions whose effect propositions have only one positive precondition and a positive effect. A similar result has been proved by Drakengren and Bjäreland [6].

The study of tractable subclasses of $\mathcal{A}$ helps also in understanding the reason of intractability. As remarked in the introduction, the language is rather simple, and its non-triviality lies only in the incomplete specification of the initial state. However, checking the consistency of a domain description made only of initially propositions is clearly polynomial: as a result, the incomplete specification of the initial state is a necessary, but not sufficient, condition for the intractability. The class of descriptions with positive effect and one positive precondition shows another source of intractability: the non-monotonicity of the set of true fluents (note, however, that the condition of having only positive preconditions and effect does not lead to tractability).

Some feature of the language that seemed, at a first look, to contribute to the intractability, in fact do not. For example, the number of actions seemed fundamental to prove NP hardness. Theorem 2 proves that it is not, since even a single action suffices to prove the problem to be intractable. The other theorems often refer to the number of actions required to obtain the NP hardness, but what is essential is that restricting our attention to domains with a constant number of actions never leads to tractability. Moreover, the actions having a large number of effects is not necessary for proving intractability.

So far for the theoretical implications of the results of the paper. Of course, the study of tractability is useful for practical implementations, but only to a certain extent: the limitations that must be imposed to the language are very restrictive. However, the NP membership of the language has also some positive effects. Any domain description expressed in $\mathcal{A}$ can be translated into a set of propositional formulas preserving consistency. This is done by translating the domain description into a set of clauses whose completion is consistent if and only if the domain is consistent. Once obtained a propositional formula expressing the considered domain, heuristics developed for sat, such as the popular GSAT, can be used.

Future work. Many dialects of $\mathcal{A}$ have been proposed in the last five years. A possible improvement of this work is the study of the computational complexity of all these formalisms. Another open problem is the "state checking", that is, given a final state σ and a sequence of actions, decide if there exists an initial state such that σ is one of the possible outcomes of the sequence. Finally, the consequences of the NP membership may be further developed. The typical approach to NP problems is that of approximation, and the significance of an approximate algorithm is related to its performance on test sets, especially when data comes from real problems.

Acknowledgments

We would like to thank Marco Schaerf for revising a draft version of this paper. Part of the work reported here has been done while the author was visiting AT&T Bell Labs. We thank Henry Kautz for his support. We also thanks Thomas Drakengren, Erik Sandewall, Michael Thielscher, and the anonymous reviewer for their comments and suggestions.

References

[1] A. Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artif. Intell.*, 49:5–23, 1991.

[2] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of IJCAI-93*, pages 866–871, 1993.

[3] C. Baral, M. Gelfond, and A. Provetti. Representing actions: Laws, observations and hypothesis. *J. of Logic Programming*, 1996. To Appear. A shorter version was presented in the 1995 Spring symposium.

[4] C. Boutilier and N. Friedman. Nondeterministic actions and the frame problem. In *Spring Symposium on Extending Theories of Action*, 1995.

[5] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, 1978.

[6] T. Drakengren and M. Bjäreland. Reasoning about actions in polynomial time. In *Proc. of IJCAI-97*, pages 1447–1452, 1997.

[7] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *J. of Logic Programming*, 17:301–322, 1993.

[8] E. Giunchiglia and V. Lifschitz. Dependent fluents. In *Proc. of IJCAI-95*, pages 1964–1969, 1995.

[9] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artif. Intell.*, 33(3):379–412, 1987.

[10] D. S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2. Elsevier, 1990.

[11] G. Kartha and V. Lifschitz. Actions with indirect effects (preliminary report). In *Proc. of KR-94*, pages 341–350. Morgan Kaufmann, 1994.

[12] R. Li and L. Pereira. What is believed is what is explained. In *Proc. of AAAI-96*, pages 550–555, 1996.

[13] F. Lin and Y. Shoham. Provably correct theories of action. *Journal of the ACM*, 42(2):293–320, 1995.

[14] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[15] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, New York, 1991.

[16] E. Sandewall. *Features and Fluents.* Oxford University Press, 1994.

[17] Marco Schaerf and Marco Cadoli. Tractable reasoning via approximation. *Artif. Intell.*, 74:249–310, 1995.

[18] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of AAAI-92*, pages 440–446, 1992.