Languages and Theories for Systematically Designing and Analyzing Autonomous Agents

Chitta Baral

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85233, USA
chitta@asu.edu

Tran Cao Son

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

October 22, 2007

Chapter 1

Introduction and Motivation

In recent years the word ‘agent’ has been used or perhaps overused with respect to many computer programs in various different contexts. In the field of artificial intelligence it has been used with qualifiers such as ‘rational’ and ‘autonomous’. In this paper we discuss a systematic approach for the design and analysis of *autonomous agents* based on recent research in reasoning about actions, goal specification and agent execution languages. To motivate our interpretation of the phrase “autonomous agents” let us start with the dictionary meanings of the terms ‘agent’ and ‘autonomy’ and interpret them in the context of Arificial Intelligence, Computer Science and Asimov’s laws of Robotics.

For the word *agent*, the American Heritage dictionary of the English language gives the meanings (i) one that acts or has the power or authority to act, (ii) one empowered to act for or represent another, (iii) a means by which something is done or caused, and (iv) a force or substance that causes a change. The Internet resource Wordnet gives the meanings (v) an active and efficient cause; capable of producing a certain effect, and (vi) a substance that exerts some force or effect

For the word *autonomous*, the American Heritage dictionary of the English language gives the meanings (i) not controlled by others or by outside forces, (ii) independent in mind or judgement or government; self directed. Wordnet gives the meanings (iii) of political bodies [syn: independent, self-governing, sovereign], (iv) existing as an independent entity and (v) of persons; free from external control and constraint in.

In our context, an artificial computer-driven agent is an entity that acts and that causes change. Since we would like to have some control over the agent, lest it makes us its slave, the qualifier ‘autonomous’ in the context of an artificial computer-driven agent means that its actions are not micro-managed by other entities, but some humans do have control over it. In other words an autonomous artificial computer-driven agent, which we will simply refer henceforth as an *autonomous agent*, can take high-level directives and goals and can figure out what actions it needs to take and execute those actions.

To be able to do this, among other things, an autonomous agent must understand high level directives, must know what actions it can do and what changes these action would cause, must have knowledge about the environment it is in and how it might change or react to its actions, must be able to observe the world (what is true when, what actions occurred when, etc.), must be able to analyze and assimilate those observations so as to make conclusions that it did not directly observe, must be able to contemplate the impact of its actions, must be able to predict how the world might evolve, must be able to contemplate what went wrong when the world does not evolve as the agent expected it to, must be able to make plans

to make the world evolve in a particular way (as directed, or so as to achieve its goal), execute those plans, and modify those plans if interfering actions beyond its control happen (i.e., the environment does not co-operate), and must be able to learn from its interaction with the environment.

In recent years a lot of research has been done with respect to each of the above aspects. In this paper we aim to connect these research into a single framework. To start with let us analyze and elaborate what technical formulations, and algorithms are needed to be able to characterize and correctly implement the various abilities mentioned in the previous paragraph.

1.1 Specifying actions and their effects

To start with we will need to specify what actions an agent can execute in what state of the world, and what its impact on the world would be. Since the number of states of the world is often exponential in the size of the properties of the world, we can not just explicitly specify a table that will tell us the transition between states of the world due to actions. In this paper we will present a brief evolution of various languages for such specification of action and their effects.

1.2 Specifying observations and figuring out the current state of the world

The specifications in the previous sub-section will tell us the transition between states due to actions and possible evolutions of the world. When an agent is situated in a world, it may have complete information (in terms of which fluents[1] are true and which are false) about a particular initial state. But after that the agent executes some actions, other actions are executed by other agents or the environment, and the agent may observe only some of the happenings and may have sensed some of the fluent values at some time points. Based on its partial knowledge the agent needs to periodically figure out its current state so as to evaluate if the world is evolving the way the agent desired it to and if not the agent can then try to change its course of actions to steer the evolution to a desired path.

Here the specification of various kinds of observations is syntactically simple, but analyzing these observations to narrow down the current state, and how the world progressed to it (from the initial state) is challenging. We will present research that have been done on this.

1.3 Expressing directives and Goals

The previous two subsections were about the specification and analysis that an agent needs to make to figure out how the world has evolved and may evolve in the future. As we mentioned earlier, we envision our 'autonomous' agent to be ready to take directives from a human. Such directives will specify *how the world should evolve*. To express such directives, as well as for the agent to express its own goals in a declarative manner we need a goal or directive specification language. Since specifying how the world should evolve means specifying the set of acceptable or desired trajectories (of the world), a starting point for a goal specification language are temporal logics, such as LTL and CTL*, that are used in program specification. Although these languages have many useful connectives such as $\Diamond$, $\Box$, $\mathcal{U}$, A, and E meaning, $eventually$, $always$, $until$, *for all paths*, and *exist path* respectively, many important

[1] By fluents we mean dynamic properties of the world.

directives can not be expressed using these languages. We will discuss recent advances in languages to express richers goals and directives.

1.4 Sensing actions and partial observability

Often an agent does not have sensors that can sense all the fluents all the time. In fact certain fluents can not be sensed directly and their values need to be inferred from other sensor values. Certain sensors can only be operated when some specific conditions are satisfied. Finally, even if an agent can sense all the fluents all the time, that may be too time consuming as well as expensive. Thus often an agent will have incomplete knowledge about the world it is in. Thus an agent needs to be able to characterize transitions between its knowledge of the world due to regular world-changing actions and sensing actions. We will discuss how to specify transitions due to sensing actions and how to characterize the difference between a world and the agent's knowledge about the world.

1.5 Control architectures and specification, synthesis and verification of control

Once we analyze an agent's ability (what actions it can do, when and what their impact would be), and are able to give a directive to an agent, the next issue is how the agent executes: what actions it executes and when. This is referred to as the control of an agent and the control program of an agent has several differences with standard computer programs.

Foremost among them is that when a basic action of an agent is performed it may change the truth value of several fluents in the world simultaneously, while similar basic steps in most[2] computer programs change the value of a single variable. This is because the variables in a program environment are not so tightly linked that changing one of them simultaneously causes change in many others. On the other hand in many of the worlds an agent resides in the fluents are tightly linked; lifting the cup lifts the water in it; moving from A to B simultaneously makes $at(A)$ false and $at(B)$ true. Some of the other differences are as follows:

- The basic actions of an agent may have non-deterministic effects, while the basic statements of a computer program usually change the program in a deterministic way.
- In most programming environments the program in focus is the only one which can change the values of the variables. In the environment of an agent other agents can change the world. Thus an agent in a dynamic environment can not just assume that the value of a fluent remains unchanged, unless the agent itself changes it.
- Automatic synthesis of agent control is an important issue with autonomous agents, while programs are normally written by people. Because of this, an agent control language depends on the assumptions about the environment, and usually the simplest one that is adequate is picked. For example, in a static domain where our agent is the only one that can make changes, and where the actions are deterministic, the agent control can be simply described by a sequence of actions. But even then automatic synthesis is difficult; its NP-complete even under simplifying assumptions.

[2] Java has object constructs that are tightly linked.

Thus the agent control language may include features whereby the agent designer may specify some domain knowledge that will help the agent synthesize its control.

Based on the above differences we need to look at agent control very differently from standard programming. To start with we need to explore control architectures ranging from reactive architecture consisting of "sense; react" cycles, deliberative architecture consisting of "observe; (re)plan; execute a bit" cycles, and various levels of hybrid architectures that allow reaction for a large number of cases, and judicious deliberation for the other cases. We then need to consider whether plans are simple action sequences, or may have sensing actions, conditionals, and more general features. From the synthesis point of view we need to explore plan synthesis algorithms for various kinds of goals, and various kinds of of plan structures. Finally, we need to consider various ways to specify domain knowledge that can help in plan synthesis. This includes procedural knowledge as in Golog, partial ordering knowledge as in HTNs and temporal knowledge specified in a temporal logic.

1.6 The reasoning issues: prediction, planning, explanation, diagnosis, plan verification, and control design

In the previous sections we discussed several different languages: languages to specify actions and their effects; languages to specify observations; languages to specify directives; and languages to specify control. Let us now give names to specifications in each of these languages and tie them together.

- Domain description D: The specification D consists of specification of actions, including sensing actions and their effects, the executability conditions of the actions, and the relationship (possibly causal) between fluents. The characterization of D is a transition function that shows the transition between states due to actions, and in presence of sensing action and incompleteness, the transition between knowledge states due to regular and sensing actions.
- Evolution description E: The evolution specification consists of specification of triggering conditions, their inhibition conditions, timing of naturally occurring actions, and ordering information between triggers. The characterization of D and E together is a set of possible trajectories.
- Observation specification O_a and O_b: Observation specifications are observations about action occurrences and value of fluents at particular time points, and ordering information between time points. Observations are grouped into two kinds: abducible observations O_a, the observations that can be explained or that are caused; and basic observations O_b, the observations that record happenings out of free will. The characterization of D, E and O_b together is a set of mapped trajectories, where each mapped trajectory consists of a trajectory and a mapping of time points, including the current time point, to states in the trajectory. We will refer to the mapped trajectories of D, E and O_b together as a *model* of (D, E, O_b). The observations in O_a are then used to select those mapped trajectories that explain O_a and filter out the remaining.
- Goal specification G: Goals are high level specification of what one desires in terms of how the world should evolve. We plan to specify it using an appropriate temporal logic. Its characterization will consist of structures made up of states, trajectories, and transition graphs, depending on the particular temporal logic used.
- Control specification C: The control specification is a control program in a particular language. In the simplest case, it will be a sequence of actions, and in more general cases it may include

features such as sensing actions, conditional statements, and non-determinism operators together with control knowledge.

- A query Q is of the form G **by_means_of** C **at_timepoint** t, where C is a control specification, G is a goal specification and t is a time point. If C is a simple sequence of actions $a_1, \ldots, a_n$ and G is a fluent f, then the query f **by_means_of** $a_1, \ldots, a_n$ **at_timepoint** t refers to enquiring if f is true in the situation obtained by executing the sequence $a_1, \ldots, a_n$ at the situation corresponding to time point t. On the other hand if G is the temporal goal $\Box f$ then the query $\Box f$ **by_means_of** $a_1, \ldots, a_n$ **at_timepoint** t refers to enquiring if f is true in all situations reached during the execution of the sequence $a_1, \ldots, a_n$ at the situation corresponding to time point t. Thus " **by_means_of** C" may refer to after C is executed or during the execution of C depending on the goal G.

- The entailments $M \models Q_1, \ldots, Q_n$ and $(D, O_b) \models Q_1, \ldots, Q_n$: $M \models Q_1, \ldots, Q_n$ is true if $Q_1 \ldots Q_n$ evaluate to true with respect to the model M. The entailment $(D, O_b) \models Q_1, \ldots, Q_n$ is true if each of the Q_is evaluate to true with respect to all the models of (D, O_b). In the entailments above queries can be replaced by observations.

We now illustrate how the models of (D, O_b) and the entailment $(D, O_b) \models Q_1, \ldots, Q_n$ is central to the notion of verifying and synthesizing appropriate controls for given directives.

- Control verification: Given D and O_b, the current time point t_n, the goal G, if we would like to verify if G is indeed obtained by means of a control C then we need to check if
  $(D, O_b) \models G$ **by_means_of** C **at_timepoint** t_n
  is true or not.

- Control generation or planning: Given a D, and O_b, the current time point t_n, the goal G, planning for the goal G starting from t_n means finding a control C such that $(D, O_b) \models G$ **by_means_of** C **at_timepoint** t_n.

- Prediction and plan verification: Given a D, E and O_b, the current time point t_n, the goal G, if we would like to verify if G is indeed obtained by means of a control C then we need to check if
  $(D, E, O_b) \models G$ **by_means_of** C **at_timepoint** t_n
  is true or not.

- Planning: Given a D, E and O_b, the current time point t_n, the goal G, planning for the goal G starting from t_n means finding a control C such that $(D, E, O_b) \models G$ **by_means_of** C **at_timepoint** t_n.

- Explanations and hypothetical explanations: Given (D, E, O_b) and abducible observations O_a, an explanation model of O_a with respect to (D, E, O_b) is a model M of (D, E, O_b) such that $M \models O_a$.

  If no such explanation model exists then we define an hypothetical explanation model of O_a with respect to (D, E, O_b) as a model M of $(D, E, O_b \cup O'_b)$ for some O'_b such that $M \models O_a$.

- Abductive Entailment and hypothetical abductive entailment: Given (D, E, O_b), abducible observations O_a, and queries $Q_1, \ldots, Q_n$:

  The abductive entailment relation $\models_a$ is defined as follows:

$\langle (D, E, O_b), O_a \rangle \models_a Q_1, \ldots, Q_n$ if O_a has at least one explanation model with respect to (D, E, O_b) and for all explanation models M of O_a with respect to (D, E, O_b), $M \models Q_1, \ldots, Q_n$.

The hypothetical abductive entailment relation $\models_a^h$ is defined as follows:

$\langle (D, E, O_b), O_a \rangle \models_a^h Q_1, \ldots, Q_n$ if O_a has at least one hypothetical explanation model with respect to (D, E, O_b) and for all hypothetical explanation models M of O_a with respect to (D, E, O_b), $M \models Q_1, \ldots, Q_n$.

- Diagnosis: Diagnosis involves finding (perhaps a minimal set of) abnormal properties of the world that explain the observations, when the assumption that everything is normal can not explain the abducible observations O_a. I.e., (D, E, O_b) which includes the normality assumption with respect to O_b does not have a model that entail O_a. In that case we need to find a modification of O_b by minimally changing the normality assumption such that the modified triplet (D, E, O_b') has at least one model that entails O_a.

- Hypothesis generation: In hypothesis generation, one is looking for hitherto unknown aspects of the world (a domain description proposition or an evolution description proposition) that will explain observations O_a that is unexplainable with respect to the currently known (D, E, O_b). If (D, E, O_b) does not have any models that entail O_a, then D', E', O_b' are possible hypothesis if $(D \cup D', E \cup E', O_b \cup O_b')$ has a model that entails O_a. One may limit and order possible hypothesis in various ways: using predefined sets from which D', E' and O_b' come from, using minimality, based on how many or what percentage of models of $(D \cup D', E \cup E', O_b \cup O_b')$ entail O_a, etc. Also, different kinds of hypothesis can be generated by restricting one or two of D', E' and O_b' to be an empty set. For example, if D' and E' are required to be empty sets then we only generate hypothetical explanations.

Chapter 2

Representing actions and the environment

If one were to start the design of an autonomous agent from scratch then a first step in the systematic design of an autonomous agent would be to analyze the kind of directives that will be given to the agent, the kind of world the agent would be in and from that design the actuators and sensors for the agent. However, often designers are given an unprogrammed artifact with built-in abilities with respect to acting and sensing and are required to develop the software that would make the artifact an autonomous agent with respect to certain kinds of directives and certain types of worlds. This means the designers would have to develop software that can synthesize controls for given directives or at the minimum can verify a given control and adapt it to new directives. Thus our first research issue becomes the representation of the world and the actions that the agent can do and the impact of these actions on the world.

Representing actions and their effects on the world has a long history in AI. The realization that the number of states of the world is often exponential with respect to the number of fluents – properties of the world, immediately ruled out a representation that stores the explicit mapping from states and actions to states. Some of the early historical landmarks on action representation include the use of STRIPS operators for planning [FN71], the invention of situation calculus [MH69], the discovery of the frame problem, the early attempt to solve the frame problem using non-monotonic logics [MH69], use of formalization of the frame problem as a benchmark for non-monotonic logics and the Yale shooting misunderstanding [McD87]. Some of the lessons from these were as follows:

1. The original STRIPS syntax is very restrictive. With slight generalization (say actions with non-deterministic effects) the semantics is no longer straightforward.

2. Representing what changes in the world due to an action is comparatively easy and succinct in a logical language. However representing what does not change, is either hard or verbose. Finding a succinct representation of what does not change is often referred to as the frame problem.

3. While non-monotonic logics can be used to for a succinct solution of the frame problem, the semantic nuances of non-monotonic logics are even difficult for specialists and need special care.

4. Representing English statements in non-monotonic logic leads to misunderstandings as the semantics of English is not precise.

Based on the above lessons the early nineties saw a large body of research on high level action description languages whose syntax were English-like and whose semantics were defined using set theory and simple

mathematical notions rather than mathematical logics. The reasoning with respect to these languages could then be done through a translation to a logical language and one now had a way to formally prove that the logical encodings were correct. The use of set theory and simple mathematical notions in defining the semantics of these languages freed the reader from knowing or learning a logical language to understand the semantics. In the rest of this section we describe several of these high level languages. Our presentation will track the evolution of these languages and will focus on the new aspects of each succeeding language. This will make the nuances of each aspect clearer as opposed to if we were to present a single language with all the features. We start with the simplest language called $\mathcal{A}$.

2.1 The language $\mathcal{A}$: motivation

The language $\mathcal{A}$ was proposed in [GL93] as a simple action description language with an English-like syntax and a semantics defined using simple set theory. Later in [Kar93] it was used to show the correctness of various logical solution of the frame problem. To illustrate the English-like syntax of $\mathcal{A}$ let us consider the description of the actions in one version of the Yale shooting problem. The description of the actions and their effects consists of the following sentences:

> *Loading the gun* causes the gun to be *loaded. Shooting* causes the turkey *to die*. *Shooting* can be executed only if the gun is *loaded*.

Using the syntax of $\mathcal{A}$ the above information is expressed as follows:

$$\begin{array}{lll} load & \textbf{causes} & loaded \\ shoot & \textbf{causes} & \neg alive \\ \textbf{executable} & shoot & \textbf{if } loaded \end{array}$$

The general syntax of the action description part of $\mathcal{A}$ is given through two kinds of constructs, effect propositions and executability conditions. They are of the following form:

- effect proposition:

$$a \textbf{ causes } l \textbf{ if } \varphi \tag{2.1}$$

- executability condition:

$$\textbf{executable } a \textbf{ if } \varphi \tag{2.2}$$

where a is an action, l is a fluent literal, and φ is a set of fluent literals or a special symbol $\top$, which is not a fluent and represents $true$. Intuitively, (2.1) says that if a is executed in a situation, in which φ holds, then l will be true in the resulting situation. (2.2) specifies the condition under which a can be executed. When $\varphi = \top$, we will drop the **if**-part in (2.1) and simply write a **causes** l.

A domain description D in $\mathcal{A}$ consists of a set of effect proposition and executability conditions. We will assume that for each action a, the executability condition **executable** a **if** $\top$ belongs to D if it does not contain any executability condition whose action is a.

Recall that an autonomous agent would often take a domain description, observations and a given goal to either validate a given control or synthesize the control. In $\mathcal{A}$ the observation language can be used to express observations of the kind: "*The turkey is initially alive.*". Using the syntax of $\mathcal{A}$ this observation is expressed as follows:

$$\textbf{initially} \ \ alive.$$

In general, observations (O) in $\mathcal{A}$ are a collection of propositions of the form:

$$\textbf{initially} \ \ l \tag{2.3}$$

where l is a fluent literal.

We often refer to the pair (D, O) where D is a domain description and O is a set of observations as an *action description*.

Queries in $\mathcal{A}$ are used to express questions such as: "Would the turkey be alive after a shoot action was attempted in the initial situation?" This is expressed in $\mathcal{A}$ as follows:

$$alive \ \textbf{after} \ \ [shoot]$$

In general a query Q in $\mathcal{A}$ is of the form:

$$l \ \textbf{after} \ \ [a_1, \ldots, a_n] \tag{2.4}$$

where l is a fluent literal and a_i's are actions. Intuitively the above query means: *Would l be true if the sequence of actions a_1, ..., a_n were attempted in the initial situation?*

For convenience, we introduce the following shorthands:

- For a sequence of literals $l_1, \ldots, l_n$,

  $$\textbf{initially} \ \ l_1, \ldots, l_n$$

  denotes the collection of observations $\{$ **initially** $\ l_i \mid 1 \leq i \leq n\}$.

- For a sequence of sets of literals $\varphi_1, \ldots, \varphi_n$,

  $$a \ \textbf{causes} \ \ l \ \textbf{if} \ \ \varphi_1 \vee \ldots \varphi_n$$

  denotes the collection of effect propositions $\{a$ **causes** $\ l$ **if** $\ \varphi_i \mid 1 \leq i \leq n\}$.

2.2 Reasoning with respect to a description in $\mathcal{A}$: Semantics of $\mathcal{A}$

The semantics of $\mathcal{A}$ defines the entailment relation $(D, O) \models Q$. In other words, given a domain description D and observation O, can we conclude the query Q from D and O. Even with the simple syntax of queries, the entailment $(D, O) \models Q$ can be used for various kinds of reasoning that an autonomous agent may do. The following example illustrates this.

Example 1 (Formulating Plan Verification) Consider the domain description[1]

$$D_1 = \left\{ \begin{array}{l} load \textbf{ causes } loaded \\ shoot \textbf{ causes } \neg alive \textbf{ if } loaded \end{array} \right\}$$

and the observation

$$O_1 = \{ \textbf{ initially } \neg loaded, alive \}.$$

If the agent's goal is to make $\neg alive$ true and it is given a possible plan of $[load, shoot]$ for achieving this goal, then the agent can verify the plan by checking whether

$$(D_1, O_1) \models \neg alive \textbf{ after } [load, shoot]$$

holds. In this case, intuitively the above entailment does hold. □

Example 2 (Formulating Simple Planning) Consider D_1 and O_1 from the previous example. Suppose the agent's goal is still to make $\neg alive$ true. However, instead of verifying a given plan, it has to find a plan for achieving his goal. In other words, the agent needs to find a sequence of actions α such that

$$(D_1, O_1) \models \neg alive \textbf{ after } \alpha$$

holds. In this case, intuitively the above entailment holds when α is $load$ followed by $shoot$, i.e., $\alpha = [load, shoot]$. □

Example 3 (Formulating Conformant Planning) Consider the domain description D_1 from Example 1 and let $O_2 = \{ \textbf{ initially } alive \}$. Suppose the agent's goal is still to make $\neg alive$ true. As in Example 2, the agent needs to find a sequence of actions α such that

$$(D_1, O_2) \models \neg alive \textbf{ after } \alpha$$

holds. Since O_2 does not completely specifies the initial situations, the plan must be able to achieve the goal no matter which of the possible initial situations is the initial situation. The planning in this situation is referred to as conformant planning. □

Example 4 (Formulating Abduction) Consider D_1 and O_2 from Example 1 and 3 respecitively; and suppose the agent wants to find out under what additional conditions (about the initial situation) he can guarantee that $\neg alive$ **after** $[load, shoot]$. This can be posed as finding O' such that

$$(D_1, O_2 \cup O') \models \neg alive \textbf{ after } \alpha$$

holds. □

We now give the semantics of the language $\mathcal{A}$.

[1] We omit **executable** $load$ **if** $\top$ and **executable** $shoot$ **if** $\top$ from D_1.

2.3 Semantics of $\mathcal{A}$

To characterize the semantics of $\mathcal{A}$ let us ponder about the information contained in D and O: O contains information about the initial state and D tells us what will be true in a state after an action is executed. In other words, D tells us about the transition between states due to actions. Moreover, to answer queries of the form

$$f \textbf{ after } [a_1, \ldots, a_n]$$

it is sufficient to know what the initial state is and the state transition due to actions. Thus the semantics of $\mathcal{A}$ involves finding the initial state σ_0 and the transition function Φ (between states due to actions) with respect to a given D and O, and using them to define the entailment between (D, O) and queries.

For a particular domain description D, we have an associated signature consisting of a set of actions Act and a set of fluents Fl. *Fluent literals* are fluents or their negations. A *state* is a subset of Fl. A fluent f holds in a state σ if $f \in \sigma$ and a fluent literal $\neg f$ holds in a state σ if $f \notin \sigma$. Given a fluent literal l, we use $\bar{l}$ to denote its negation. In other words, if $l = f$ for some fluent f then $\bar{l} = \neg f$ and if $l = \neg f$ for some fluent f then $\bar{l} = f$. An action a is *executable* in a state σ if D contains an executable condition **executable** a **if** φ and φ holds in σ.

Definition 1 A set of fluents σ is an *initial state* corresponding to O, if for all observations of the form **initially** l (where l is a fluent literal), l holds in σ.

The transition function of a domain description is defined next.

Definition 2 Given a domain description D in $\mathcal{A}$ its *transition function* is any function Φ from states and actions to states that satisfies the following conditions:

For all actions a, fluents f, and states σ:

- if a is executable in σ then
  - If a **causes** f **if** φ belongs to D and φ holds in σ then f must be in $\Phi(a, \sigma)$.
  - If a **causes** $\neg f$ **if** φ belongs to D and φ holds in σ then f must not be in $\Phi(a, \sigma)$.
  - If D does not include such effect propositions (about a and f) then $f \in \Phi(a, \sigma)$ iff $f \in \sigma$.
- if a is executable in σ then $\Phi(a, \sigma)$ is undefined. □

Proposition 1 For any domain description D, there is at most one transition function.

Proof. Assume that D has two distinct transition function Φ and Ψ. This means that there exists some state s and action a such that $\Phi(a, \sigma) \neq \Psi(a, \sigma)$. Clearly, neither $\Phi(a, \sigma)$ nor $\Psi(a, \sigma)$ can be undefined. This means that there exists some $f \in Fl$ such that $f \in \Phi(a, \sigma)$ and $f \notin \Psi(a, \sigma)$. $f \in \Phi(a, \sigma)$ implies that (i) there exists some a **causes** f **if** φ in D such that φ holds in σ or (ii) $f \in \sigma$. (i) would indicate that $f \in \Psi(a, \sigma)$ and hence cannot happen. (ii), together with $f \notin \Psi(a, \sigma)$, implies that there exists some a **causes** $\neg f$ **if** ψ in D such that ψ holds in σ. But this means that f must not be in $\Phi(a, \sigma)$. A contradiction with $f \in \Phi(a, \sigma)$, i.e., a contradiction with our assumption that D has two distinct transition functions. □

Definition 3 A domain description D is said to be *consistent* if it has a transition function.

Remark 1 *Following Proposition 1, a consistent domain description has a unique transition function. For convenience, the transition function of a consistent domain description D will be denoted by Φ_D.*

The next example shows that not every domain description is consistent.

Example 5 (Inconsistent Domain Description) it is easy to see that the domain description with two effect propositions

$$a \textbf{ causes } f \textbf{ if } h$$
$$a \textbf{ causes } \neg f \textbf{ if } g$$

is inconsistent since it does not have any transition function Φ since for the state $\sigma = \{h, g\}$ and action a, $\Phi(a, \sigma)$ is not defined. □

Remark 2 *Inconsistent domain descriptions indicate that there is something wrong and/or something missing in the modeling process. (Need more elaboration ???)*

Proposition 2 For any consistent domain description D, its transition function Φ_D satisfies the condition:

- if a is executable in σ then

  $$\Phi_D(a, \sigma) = \sigma \cup E^+_D(a, \sigma) \setminus E^-_D(a, \sigma)$$

  where

  $$E^+_D(a, \sigma) = \{f \mid a \textbf{ causes } f \textbf{ if } \varphi \in D \text{ and } \varphi \text{ holds in } \sigma\}$$

  and

  $$E^-_D(a, \sigma) = \{f \mid a \textbf{ causes } \neg f \textbf{ if } \varphi \in D \text{ and } \varphi \text{ holds in } \sigma\}.$$

- if a is not executable in σ then $\Phi_D(a, \sigma)$ is undefined.

Proof. Follows directly from Definition 2. □

Example 6 Consider the action description (D_1, O_1) from Example 1. Let

$$\sigma_0 = \{alive\}.$$

We have that

$$\Phi_{D_1}(shoot, \sigma_0) = \{alive\}$$

because $E^+_{D_1}(shoot, \sigma_0) = E^-_{D_1}(shoot, \sigma_0) = \emptyset$ and

$$\Phi_{D_1}(load, \sigma_0) = \{alive, loaded\}$$

since $E^+_{D_1}(load, \sigma_0) = \{loaded\}$ and $E^-_{D_1}(load, \sigma_0) = \emptyset$ □

It is easy to see that if for every fluent f, $\{$ **initially** f, **initially** $\neg f\} \not\subseteq O$, then there exists at least one initial state corresponding to O. We wills say that O is *consistent* if there exists at least one initial state corresponding to O.

Definition 4 For a transition function Φ, a state σ, and a sequence of actions $a_1, \ldots, a_n$, we define the extended transition function of Φ, denoted by $\widehat{\Phi}$, as follows.

$$\widehat{\Phi}([a_1, \ldots, a_n], \sigma) = \begin{cases} \sigma & \text{if } n = 0 \\ \Phi(a_n, \widehat{\Phi}([a_1, \ldots, a_{n-1}], \sigma) & \text{otherwise} \end{cases}$$

Definition 5 (σ_0, Φ) is a model of (D, O) iff Φ is the transition function of D and σ_0 is an initial state corresponding to O.

Definition 6 Let (D, O) be an action description.

- (D, O) is said to be *consistent* if it has a model.
- (D, O) is said to be *complete* if it has a unique model.
- For an action sequence α, $(D, O) \models f$ **after** α if for all models (σ_0, Φ) of (D, O), f holds in $\widehat{\Phi}_D(\alpha, \sigma_0)$.
- An action sequence α is a *plan* for a fluent formula φ if $(D, O) \models \varphi$ **after** α.

Observe that consistency of an action description requires the consistency of its domain description and observations. Because of the uniqueness of the transition function, completeness is equivalent to the uniqueness of the initial state. This means that for each $f \in Fl$, $\{$ **initially** f, **initially** $\neg f\} \cap O \neq \emptyset$.

Example 7 Continuing with Example 6, we have that the description (D_1, O_1) is complete and has a unique initial state $\sigma_0 = \{alive\}$. Furthermore, $\Phi_{D_1}(load, \sigma_0) = \{loaded, alive\}$.

Let $\sigma_1 = \{loaded, alive\}$. We have that

$$E^+_{D_1}(shoot, \sigma_1) = \emptyset \quad \text{and} \quad E^-_{D_1}(shoot, \sigma_1) = \{alive\}.$$

Hence, $\widehat{\Phi}([load, shoot], \sigma_0) = \Phi(shoot, \sigma_1) = \{loaded\}$. We have that $\neg alive$ holds in $\{loaded\}$, i.e.,

$$(D_1, O_1) \models \neg alive \textbf{ after } [load, shoot].$$

□

Exercise 1 *Let* $D'_1 = D_1 \cup \{shoot$ **causes** $\neg loaded\}$. *Prove that*

$$(D'_1, O_1) \models \neg alive \textbf{ after } [load, shoot].$$

2.4 Complexity of reasoning and planning in $\mathcal{A}$

$\mathcal{A}$ is among the simples languages for the specification of actions and their effects. In this section we analyze the complexity of reasoning and planning in this simple langauge. To this end, we say that the size of a domain description D in the language $\mathcal{A}_0$ is the sum of the number of actions, the number of fluents, and the number of propositions of the form (2.1) and denote it by $|D|$. We consider the following decision problems:

- C1 (*Consistent domain*) Given a domain description D, determine whether D is consistent or not.
- CI (*Consistent initial state*) Given a consistent domain description D, a set of observations O, and a set of fluents σ_0, determine whether σ_0 is an initial state with respect to (D, O).
- PC (*Planning with complete initial state*) Given a consistent domain description D, a complete set of observations about the initial state O, and a fluent literal l, a polynomial function $f : N \rightarrow N$, determining an action sequence $a_1, \ldots, a_m$, where m is bounded by $f(|D|)$, such that $(D, O) \models l$ **after** $[a_1, \ldots, a_m]$.

Theorem 1 **1.** *C1 is in P.*

2. *CI is in P.*

3. *PC is NP-complete.*

Proof.

1. Φ_D exists iff for every pair of effect propositions of the form

$$a \textbf{ causes } f \textbf{ if } \varphi \quad \text{and} \quad a \textbf{ causes } \neg f \textbf{ if } \psi,$$

there exists no state σ such that both φ and ψ hold in σ. Due to the consistency of a state, this condition is satisfied iff there exists a fluent literal l such that $l \in \varphi$ and $\bar{l} \in \psi$, i.e., $\varphi \cap \bar{\psi} \neq \emptyset$, where $\bar{\delta} = \{\bar{l} \mid l \in \delta\}$. Clearly, this can be checked in polynomial time (in the size of the number of fluents). The conclusion follows from the fact that there are less than $|D|^2$ pairs of effect propositions.

2. Since O consists of observations of the form **initially** l, it has at most $2 * |Fl|$ propositions where $|Fl|$ is the number of fluents in Fl. Deciding whether or not σ_0 satisfies **initially** l is equivalent to checking whether $l \in \sigma_0$ (if $l \in Fl$) or $l \notin \sigma_0$ (if $l = \neg f$ for some $f \in Fl$). This can be done in at most $|Fl|$ operations. Thus, the problem is polynomial in the size of the number of fluents of D.

3. Given (D, O), l, and f, let $\sigma_0 = \{f \mid f$ is a fluent and **initially** f belongs to $O\}$. Since O is complete, σ_0 is unique. Furthermore, it is easy to see that given a state σ and an action a, computing $\Phi_D(a, \sigma)$ is polynomial. As such, computing $\widehat{\Phi}_D([a_1, \ldots, a_m], \sigma_0)$ is also polynomial if m is bounded by $f(|D|)$.

(i) **Membership:** It is easy to see that the following guess-and-check NP-algorithm can be used to solve the PC problem.

- *Guess*: a sequence of actions $a_1, \ldots, a_m$ where m is bounded by $f(|D|)$.
- *Check*: whether l holds in $\widehat{\Phi}_D([a_1, \ldots, a_m], \sigma_0)$.

This indicates that PC belongs to the class of NP problems.

(ii) **Hardness:** We now reduce 3SAT to this problem. Let F be the formula

$$(f_1^1 \vee f_1^2 \vee f_1^3) \wedge \ldots \wedge (f_n^1 \vee f_n^2 \vee f_n^3).$$

Let $g_1, \ldots, g_n, g$ be new propositions not in the language of F. We construct (D, O) as follows.

- The language of D consists of actions a, a', and a_p for each proposition p in the language of F. D consists of the following effect propositions:
  - For each i,
    $$a \textbf{ causes } g_i \textbf{ if } f_i^1$$
    $$a \textbf{ causes } g_i \textbf{ if } f_i^2$$
    $$a \textbf{ causes } g_i \textbf{ if } f_i^3$$
    belong to D.
  - D contains the effect proposition
    $$a' \textbf{ causes } g \textbf{ if } g_1, \ldots, g_n$$
  - For each proposition p in the language of F, D contains the effect propositions
    $$a_p \textbf{ causes } p \textbf{ if } \neg p$$
    and
    $$a_p \textbf{ causes } \neg p \textbf{ if } p.$$
- O consists of the following observations:
  - **initially** $\neg p$ for every proposition p in the language of F, and
  - **initially** $\neg g_1, \ldots, \neg g_n, \neg g$.
- The goal of the planning problem is g.

Let α be an action sequence such that $(D, O) \models g$ **after** α. We can observe the following:

- α must contain the actions a and a' and a must occur before a'. (Because a' is the only action for achieving g. Furthermore, for g to be true after the execution of a', $g_1, \ldots, g_n$ need to be true and this can only be achieved by a.)
- $(D, O) \models g$ **after** β where β is obtained from α by (i) removing all the action occurrences after the occurrence of a' and (ii) removing all the action occurrences between a and a'.

The above observations, together with the fact that a_p flips the truth value of p, allow us to conclude that there exists a sequence of actions $[a_1, \ldots, a_m]$ such that

$$(D, O) \models g \textbf{ after } [a_1, \ldots, a_m]$$

iff there exists a sequence of actions $b_1, \ldots, b_k, a, a'$ where $k \leq n_p$ (n_p is the number of propositions in the language of F) and $b_i \neq b_j$ for every pair of i and j, $1 \leq i \neq j \leq n$ such that

$$(D, O) \models g \textbf{ after } [b_1, \ldots, b_k, a, a'].$$

Furthermore, let I be a truth value assignment for propositions in the language of F. Assume that $p_1, \ldots, p_k$ are true and other propositions are false w.r.t. I. It is easy to check that $(D, O) \models g$ **after** $[a_{p_1}, \ldots, a_{p_k}, a, a']$ iff I is a model of F. This one-to-one correspondence between interpretations of F and plans of length less than or equal to $n_p + 2$ for g shows that F is satisfiable iff there exists a plan for g with respect to (D, O). □

2.5 Implementing reasoning and planning in $\mathcal{A}$ by translating to SAT

In this section we show how we can encode domain descriptions of $\mathcal{A}_0$ in SAT and use model finding and reasoning in SAT to reason and plan with respect to $\mathcal{A}_0$. We will address the following problems:

- **Hypothetical reasoning**: Given a domain description D, a complete set of observations O, a fluent literal l, and an action sequence $[a_1, \ldots, a_t]$, checking whether or not $(D, O) \models l$ **after** $[a_1, \ldots, a_t]$ holds.
- **Planning with complete information**: Given a domain description D, a complete set of observations O, a fluent literal l, an interger t, checking whether or not there exists an action sequence $[a_1, \ldots, a_t]$ such that $(D, O) \models l$ **after** $[a_1, \ldots, a_t]$.

2.5.1 Encoding for hypothetical reasoning

Given an integer n, we construct a theory $tr\_do(D, O, n)$ for reasoning about effects of action sequences which contains at most n actions. The alphabet of $tr\_do(D, O, n)$ consists of propostion of the form $f[i]$ and $a[i]$ where f is a fluent, a is an action, and i is an integer between 1 and $n + 1$. Intuitively, $tr\_do(D, O, n)$ encodes the evolution of the world through the execution of the action sequence. Intuitively, $f[i]$ (resp. $\neg f[i]$) encodes that f is true (resp. false) in the state after the execution of the first $i - 1$ actions of the sequence. Similarly, $a[i]$ (resp. $\neg a[i]$) states that the action a is executed after the execution of the first $i - 1$ actions of the sequence. We introduce the following notations.

- For each fluent f, let
  - Act_f^+ be the collection of effect propositions of the form a **causes** f **if** φ in D; and
  - Act_f^- be the collection of effect propositions of the form a **causes** $\neg f$ **if** φ in D.
- For a set of fluent literals φ and an integer i,
  - $\varphi[i]$ denotes the formula $\bigwedge_{l \in \varphi} l[i]$; and
  - $\neg\varphi[i]$ denotes the formula $\bigvee_{l \in \varphi} \bar{l}[i]$. (Recall that $\bar{l}$ denotes the negation of l.)

The theory $tr\_do(D, O, n)$ is defined as follows.

- For each fluent f and an integer i, $1 \leq i \leq n$, the formula

$$f[i{+}1] \Leftrightarrow \left(\bigwedge_{a \textbf{ causes } f \textbf{ if } \varphi \in Act_f^+} (a[i] \wedge \varphi[i]) \right) \vee \left(f[i] \wedge \bigwedge_{a \textbf{ causes } \neg f \textbf{ if } \varphi \in Act_f^-} (\neg a[i] \vee \neg\varphi[i]) \right) \tag{2.5}$$

  belongs to $tr\_do(D, O, n)$.
- For each **initially** f in O, the proposition $f[1]$ is in $tr\_do(D, O, n)$.
- If $a_1, \ldots, a_r$ are all the actions in the domain, then

  - for all $1 \leq i \leq n$, if **executable** a **if** φ belongs to D and $\varphi \neq \top$, $a[i] \Rightarrow \varphi[i]$ belongs to $tr\_do(D, O, n)$.
  - for all $1 \leq i \leq n$, the formula $a_1[i] \vee \ldots \vee a_r[i]$ is in $tr\_do(D, O, n)$.
  - for all $1 \leq i \leq n$ and $1 \leq j, k \leq r$ where $j \neq k$, the formula $\neg(a_j[i] \wedge a_k[i])$ is in $tr\_do(D, O, n)$.

- A query Q given by f **after** $a_1, \ldots, a_t$ is translated to the formula

$$a_1[1] \wedge \ldots \wedge a_t[t]$$

  which we will refer to as $tr\_q(Q)$.

The following proposition is about the correctness of the above translation.

Proposition 3 Given a consistent domain description D, a complete set of initial state observations O and a query Q of the form l **after** $a_1, \ldots, a_t$, $(D, O) \models Q$ iff $tr\_do(D, O, t) \cup tr\_q(Q) \models l[t+1]$.

Proof. Let M be a model of $tr\_do(D, O, t)$ and $s_i(M) = \{f \mid f \in Ft,\ f[i] \text{ is true in } M\}$. The encoding of $tr\_do(D, O, t)$ ensures that for each integer i, $1 \leq i \leq t$, there exists a unique action a such that $a[i]$ is true in M and if $a[i]$ is true in M then a is executable in $s_i(M)$. We first prove that $s_1(M) = \sigma_0$ where σ_0 be the initial state of (D, O). Since O is complete, for every fluent $f \in Ft$, we have that

- $f \in \sigma_0$ iff **initially** f belongs to O iff $f[1]$ belongs to $tr\_do(D, O, t)$ iff $f \in s_1(M)$.
- $f \notin \sigma_0$ iff **initially** $\neg f$ belongs to O iff $\neg f[1]$ belongs to $tr\_do(D, O, t)$ iff $f \notin s_1(M)$.

The above two items conclude that $s_1(M) = \sigma_0$.

We will now prove that $s_{i+1}(M) = \Phi(a_i, s_i(M))$ for $1 \leq i \leq t+1$ by induction over i where $a_i[i]$ is true in M.

- **Base**: The construction of $tr\_do(D, O, t)$ implies that there exists some action a_1 such that $a_1[1]$ is true in M and a_1 is executable in $s_1(M)$. Thus $\Phi(a_1, s_1(M))$ is defined. Let f be a fluent. We have that $f \in \Phi(a_1, s_1(M))$ if one of the following two conditions is satisfied.

  (i) $f \in E_D^+(a_1, s_1(M))$. This implies that there exists an effect proposition a_1 **causes** f **if** φ in Act_f^+ such that φ is true in $s_1(M)$. By Equation 2.5, we have that $f[2]$ is true in M, i.e., $f \in s_2(M)$.

  (ii) $f \in s_1(M)$ and $f \notin E_D^-(a_1, s_1(M))$. This implies that $f[1]$ is true in M and for every effect proposition b **causes** $\neg f$ **if** φ in Act_f^-, either $b \neq a_1$, and hence, $\neg b[1]$ is true in M or $b = a_1$, and hence, φ does not hold in $s_1(M)$, i.e., $\neg\varphi[1]$ is true in M. Therefore, $f[1] \wedge \bigwedge_a$ **causes** $\neg f$ **if** $_{\varphi \in Act_f^-}(\neg a[1] \vee \neg\varphi[1])$ is true in M, and hence, $f[2]$ is true in M, i.e., $f \in s_2(M)$.

- **Step**: The proof for the inductive step is similar to the base case and is omitted here for brevity.