Chapter 5

Planning problems

Currently, we can find different definitions for Planning. For instance, McCarthy defines *planning* in [25] as the restricted problem of finding a finite sequence of actions that will achieve a *goal*, where goal is usually specified as one or a set of world states. In this work we are interested in some formulations that attempt to solve the planning problem based on Logic-based approaches. In this chapter we start presenting an overview about the Logic-based Planning approaches. Then, we present an overview about one of the AI planning approaches to model and solve planning problems in a natural and elegant way is *Answer Set Planning*.

In this chapter we also study Language $\mathcal{PP}$ [43] that allows one to specify preferences among feasible plans and temporal preferences over plans too. Moreover in Contribution Part of this work we present an extension for Language $\mathcal{PP}$ and we show how this language and its extension can be applied to obtain the preferred evacuation plans.

For more information about planning, [11] outlines some more classical planning work, [41] covers most of the advances in the last 15 years in planning, and [29] presents a summary of classical planning techniques. Finally, it is important to mention that *planning* is deciding what to do, while *scheduling* is deciding when to do it.

5.1 Planning

Logic-based Planning is also sometimes categorized as change-based planning. It is based on an initial state, a set of actions, a goal, a planner and an executor [17]. Then according to the initial state, the *planner* will try to generate a plan such that when this plan is executed by the *executor* will result in the state g satisfying the goal state description.

This definition of Logic-based Planning lead us into the definition of *Situation Calculus* introduced in [26]. Situation Calculus is the most studied formalism for doing causal reasoning. A *situation* is in principle a snapshot of the world at an instant. One never knows a situation —one only knows facts about a situation. Events occur in situations and give rise to new situations. There are many variants of situation calculus, and none of them has come to dominate. Functions of situations in situation calculus are defined in terms of *fluents* [25]. The simplest fluents are propositional and have truth values. There are also fluents with values in numerical or symbolic domains. Situational fluents take on situations as values.

Some of the different planning problems are the following:

—*Frame Problem*[25]. This is the problem of how to express the facts about the effects of actions and other events in such a way that it is not necessary to explicitly state for every event, the fluents it does not affect. For instance, traveling from Mexico to Lyon does not affect the number of arms I have.

—*Qualification Problem* [25]. This concerns how to express preconditions for actions and other events. For instance, it is necessary to have a ticket to fly on a commercial airplane and it is also necessary to have money to buy the ticket, etc.

—*Ramification Problem*[25]. Events often have other effects than those we are immediately inclined to put in the axioms concerned with the particular kind of event.

For instance, if I travel from Mexico to Lyon, then as a result each part of the airplane is now at Lyon, so the airplane captain, etc.

It is worth mentioning at this point, that planning is a key ability for intelligent systems, increasing their autonomy and flexibility through the construction of sequences of actions to achieve their goals. Planning has been an area of research in artificial intelligence for over three decades and it is called *AI planning*. Planning techniques have been applied in a variety of tasks including robotics, process planning, web-based information gathering, autonomous agents and spacecraft mission control. AI planning is action-based where a declarative description of actions is used. Additionally, AI planning specifies declaratively what the operators are able to do and it is not prescribed how to perform complex tasks, since the planner must infer that, using trial-and-error search.

Currently, one of the AI planning approaches to model and solve planning problems in a natural and elegant way is *Answer Set Planning*. We will study this approach in the following Section.

5.2 Answer Set Planning

In a planning problem, we are interested in looking for a sequence of actions that leads from a given initial state to a given goal state. To specify a planning problem completely, we need to explicit which actions are allowed in a plan. The key element of answer set planning is the representation of a dynamic domain in the form of a "history program"—a program whose answer sets represent possible "histories", or evolutions of the system, over a fixed time interval.

Currently, there exist different action languages that are formal models of parts of the natural language that are used for talking about the effects of actions and in

particular they are used to to model planning problems [16]. Some of these action languages are $\mathcal{A}, \mathcal{B}$, and $\mathcal{C}$ [16]. Moreover, a planning problem specified in these kinds of languages has a natural and easy encoding in Answer Set Programming. In this Section we shall present a brief overview extracted from [4] about language $\mathcal{A}$ and $\mathcal{B}$, the use of these languages to specify planning problems and the encoding of planning problems in Answer Set Programming.

5.2.1 Language $\mathcal{A}$

The alphabet of the language $\mathcal{A}$ consists of two nonempty disjoint sets of symbols **F** and **A**. They are called the set of fluents, and the set of actions. Intuitively, a fluent expresses the property of an object in a world, and forms part of the description of states of the world. A *fluent literal* is a fluent or a fluent preceded by $\sim$. A *state* σ is a set of fluents. We say a fluent f holds in a state σ if $f \in \sigma$. We say a fluent literal $\sim f$ holds in σ if $f \notin \sigma$. Actions when successfully executed change the state of the world. Situations are representations of the history of action execution. In the initial situation $[a_n, \ldots, a_1]$ corresponds to the history where action a_1 is executed in the initial situation, followed by a_2, and so on until a_n. There is a simple relation between situations and states. In each situation s some fluents are true and some others are false, and this 'state of the world' is the state corresponding to s.

The language $\mathcal{A}$ can be divided in three sub-languages: *Domain description language*, *Observation language*, and *Query language*. We are going to present an overview about these sublanguages, the details can be founded in [16, 4].

- *Domain description language.* It is used to express the transition between states due to actions. The domain description D consists of effect propositions of the

following form:

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r \tag{5.1}$$

where a is an action, f is a fluent literal, and $p_1, \ldots, p_n, q_1, \ldots, q_r$ are fluents. Intuitively, the above effect proposition means that if the fluent literals $p_1, \ldots, p_n$, $\sim q_1, \ldots, \sim q_r$ hold in the state corresponding to a situation s then in the state corresponding to the situation reached by executing a in s the fluent literal f must hold. The role of effect propositions is to define a transition function from states and actions to states. Given a domain description D, such a transition function Φ should satisfy the following properties. For all actions a, fluents g, and states σ:

- if D includes an effect proposition of form (5.1) where f is the fluent g and $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ hold in σ then $g \in \Phi(a, \sigma)$;
- if D includes an effect proposition of form (5.1) where f is a negative fluent literal $\sim g$ and $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ hold in σ then $g \notin \Phi(a, \sigma)$; and
- if D does not include such effect proposition, then $g \in \Phi(a, \sigma)$ iff $g \in \sigma$.

- *Observation language.* A set of observations O consists of value propositions of the following form:

$$f \textbf{ after } a_1, \ldots, a_m \tag{5.2}$$

where f is a fluent literal and $a_1, \ldots, a_m$ are actions. Intuitively, the above value proposition means that if $a_1, \ldots, a_m$ would be executed in the initial situation then in the state corresponding to the situation $[a_m, \ldots, a_1]$, f would hold.

When $a_1, \ldots, a_m$ is an empty sequence, we write the above as follows:

$$\textbf{initially } f \tag{5.3}$$

In this case the intuitive meaning is that f holds in the state corresponding to the initial situation. Given a consistent domain description D the set of observations

O is used to determine the states corresponding to the initial situation, referred to as *initial states* and denoted by σ_0.

- *Query language.* Queries also consist of value propositions of the form (5.2). We say a consistent domain description D in the presence of a set of observations O entails a query Q of the form (5.2) if for all initial states σ_0 corresponding to (D, O), the fluent literal f holds in the state $[a_m, \ldots, a_1]\sigma_0$. We denote this as $D \models_O Q$.

Language $\mathcal{A}$ also can be extended to allow *executability conditions* and *static causal propositions* in the domain description part. Intuitively they indicate if an action is executable in a given state. An *executability condition* has the following form:

$$\textbf{executable } a \textbf{ if } p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r \tag{5.4}$$

where a is an action and, $p_1, \ldots, p_n, q_1, \ldots, q_r$ are fluents. Intuitively, the above executability condition means that if the fluent literals $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ hold in the state σ of a situation s, then the action a is executable in s.

In particular, the language $\mathcal{A}$ can be used to model planning problems. In planning a domain description D, a set of observations about the initial state O, and a collection of fluent literals $G = \{g_1, \ldots, g_l\}$, which we will refer to as a goal are given. We are required to find a sequence of actions $a_1, \ldots, a_n$ such that for all $1 \leq i \leq l$, $D \models_O g_i$ **after** $a_1, \ldots, a_n$. We then say that $a_1, \ldots, a_n$ is a *plan* for goal G (or achieves goal G) with respect to (D, O). Given a plan $a_1, \ldots, a_n$ for a goal G with respect to (D, O), we also say that the sequence $\sigma_0, a_1, \sigma_1, \ldots, a_n, \sigma_n$ where σ_0 is the initial state and $\sigma_i = \Phi_D(a_i, \sigma_{i-1})$ with $1 \leq i \leq n$ is a *history* of the transition function Φ_D.

The following example shows how to specify the simple blocks world planning problem in language $\mathcal{A}$ that it is presented in [4].

Example 5.1. The simple blocks world planning problem in language $\mathcal{A}$:

1. Sort: $block(X)$. In the rest of this example, X and Y range over the sort $block$.
2. Fluents: $on(X,Y), ontable(X), clear(X), holding(X), handempty$
3. actions: $pick\_up(X), put\_down(X), stack(X,Y), unstack(X,Y)$
4. Executability conditions, denoted by D:

   executable $pick\_up(X)$ **if** $clear(X), ontable(X), handempty.$

   executable $put\_down(X)$ **if** $holding(X).$

   executable $stack(X,Y)$ **if** $holding(X), clear(Y).$

   executable $unstack(X,Y)$ **if** $clear(X), on(X,Y), handempty.$
5. Effect propositions.

   $pick\_up(X)$ **causes** $\sim ontable(X), \sim clear(X), holding(X), \sim handempty.$

   $put\_down(X)$ **causes** $ontable(X), clear(X), \sim holding(X), handempty.$

   $stack(X,Y)$ **causes** $\sim holding(X), \sim clear(Y), clear(X), handempty, on(X,Y).$

   $unstack(X,Y)$ **causes** $holding(X), clear(Y), \sim clear(X), \sim handempty, \sim on(X,Y).$
6. Considering that the sort blocks as $\{a,b,c,d\}$ the initial conditions, denoted by I, are described as:

   $initially(clear(a)).$

   $initially(clear(c)).$

   $initially(clear(d)).$

   $initially(ontable(c)).$

   $initially(ontable(b)).$

   $initially(on(a,b)).$

   $initially(on(d,c)).$

   $initially(handempty).$
7. The goal conditions, denoted by G, are the set of literals $\{on(a,c), on(c,b), ontable(d)\}$.

□

5.2.2 Answer set encoding of planning problems modeled in language $\mathcal{A}$

Given a planning problem expressed in an action language by means of a triple (D, O, G) such that D is a domain description, O is a set of observations and G is the set of goal conditions, it is possible to define different answer set encodings of it. In this section we present one of them that it is described in [4], denoted as $\pi(D, O, G, l)$ where l is a positive integer corresponding to the length of the plan, i.e., the number of actions that must be performed to achieve the goal. Then, it is possible to obtain the solution of the planning problem (the plans) from the answer sets of $\pi(D, O, G, l)$. The basic idea is that we decide on a plan length l to enumerate action occurrences up to time points corresponding to that length, and to encode goals as constraints about the time point $l + 1$, such that possible answer sets that encode action occurrences that do not satisfy the goal at time point $l + 1$ are eliminated. Thus each of the answer sets which survived the constraints encodes a plan. Then, given a domain description D, a set of observations O, a plan length l, and the goal conditions G, we construct the following program $\pi(D, O, G, l)$. The encoding $\pi(D, O, G, l)$ is divided in two parts: the *domain dependent part*, denoted as $\pi^{dep}(D, O, G, l)$, and the *domain independent part*, denoted as $\pi^{indep}(D, O, G, l)$. The former is a direct translation of the domain description in language $\mathcal{A}$. The latter is independent of the domain and may be used for planning with other domains.

It is important to mention that we have two sorts (see Section 4.1): *fluents* and *actions*, and variables of these sorts are denoted by $A, A', \ldots, F, F', \ldots$ respectively. The sorts of the arguments of function symbols and predicates are clear from the context.

We are going to present the answer set encoding given in [4] about the simple blocks world planning problem from Example 5.1 expressed in an action language $\mathcal{A}$.

It is important to mention that in the answer set encodings of planning problems expressed in action language $\mathcal{A}$ that we present in this work, if f is a negative fluent literal $\sim g$ then f will be encoded as $neg(g)$. Moreover, $neg(g)$ corresponds to the strong or classical negation as we will see in the complete encoding when we present the set of axioms that define $neg(g)$.

The answer set encoding is defined as follows:

1. *The Domain dependent part* $\pi^{dep}(D, O, G, l)$: This consists of five parts, defining the domain, the executability conditions, the dynamic causal laws, the initial state, and the goal condition.

   (a) *The domain* $\pi^{dep.dom}(D)$: This part defines the objects in the world, the fluents and the actions.

   For instance:

   $block(a).$ $block(b).$ $block(c).$ $block(d).$

   $fluent(on(X,Y)) \leftarrow block(X), block(Y).$

   $fluent(ontable(X)) \leftarrow block(X).$

   $fluent(clear(X)) \leftarrow block(X).$

   $fluent(holding(X)) \leftarrow block(X).$

   $fluent(handempty).$

   $action(pickup(X)) \leftarrow block(X).$

   $action(putdown(X)) \leftarrow block(X).$

   $action(stack(X,Y)) \leftarrow block(X), block(Y).$

   $action(unstack(X,Y)) \leftarrow block(X), block(Y).$

   (b) *Translating executability conditions* $\pi^{dep.exec}(D)$: Here, $exec(a, p)$ means that p is among the executability conditions of a, and intuitively, a is executable in a state if all its executability conditions hold in that state. The later part will be

encoded as a domain independent rule. Hence, for every executability condition of the form (5.4), i.e., **executable** a **if** $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ $\pi^{dep.exec}(D)$ contains a the following set of rules:

$$exec(a, p_1).$$
$$exec(a, p_1)$$
$$\vdots$$
$$exec(a, p_n).$$
$$exec(a, neg(q_1)).$$
$$\vdots$$
$$exec(a, neg(q_r)).$$

For instance:

$exec(pick\_up(X), clear(X)).$

$exec(pick\_up(X), ontable(X)).$

$exec(pick\_up(X), handempty).$

$exec(put\_down(X), holding(X)).$

$exec(stack(X, Y), holding(X)).$

$exec(stack(X, Y), clear(Y)).$

$exec(unstack(X, Y), clear(X)).$

$exec(unstack(X, Y), on(X, Y)).$

$exec(unstack(X, Y), handempty).$

(c) *Translating effect propositions* $\pi^{dep.dyn}(D)$: The effect propositions in D are translated as follows. For every effect proposition of the form (5.1), i.e., a **causes** f **if** $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ if f is a fluent then $\pi^{dep.dyn}(D)$ contains the following rule:

$$causes(a, f).$$

else, if f is the negative fluent literal $\sim g$ then $\pi^{dep.dyn}(D)$ contains the following rule:

$$causes(a, neg(g)).$$

For instance:

$causes(pick\_up(X), neg(ontable(X))).$

$causes(pick\_up(X), neg(clear(X))).$

$causes(pick\_up(X), holding(X)).$

$causes(pick\_up(X), neg(handempty)).$

$causes(put\_down(X), ontable(X)).$

$causes(put\_down(X), clear(X)).$

$causes(put\_down(X), neg(holding(X))).$

$causes(put\_down(X), handempty).$

$causes(stack(X, Y), neg(holding(X))).$

$causes(stack(X, Y), neg(clear(Y))).$

$causes(stack(X, Y), clear(X)).$

$causes(stack(X, Y), handempty).$

$causes(stack(X, Y), on(X, Y)).$

$causes(unstack(X, Y), holding(X)).$

$causes(unstack(X, Y), clear(Y)).$

$causes(unstack(X, Y), neg(clear(X))).$

$causes(unstack(X, Y), neg(handempty)).$

$causes(unstack(X, Y), neg(on(X, Y))).$

(d) *Translating observations* $\pi^{dep.init}(O)$*:* This part defines the initial state by explicitly listing which fluents are true in the initial state. It is assumed that the fluents not explicitly listed to be true are false in the initial state. Thus the

knowledge about the initial state is assumed to be complete. For every value proposition of the form (5.2), i.e., **initially** f if f is a fluent then $\pi^{dep.init}(O)$ contains the following rule:

$$initially(f).$$

else, if f is the negative fluent literal $\sim g$ then $\pi^{dep.init}(O)$ contains the following rule:

$$initially(neg(g)).$$

For instance:

$initially(handempty).$

$initially(clear(a)).$

$initially(clear(d)).$

$initially(ontable(c)).$

$initially(ontable(b)).$

$initially(on(a, b)).$

$initially(on(d, c)).$

(e) *Goal conditions* $\pi^{dep.goal}(G)$*:* This part lists the fluent literals that must hold in a goal state. For every fluent literal in G, if f is a fluent then $\pi^{dep.goal}(G)$ contains the following rule:

$$finally(g).$$

else, if f is the negative fluent literal $\sim g$ then $\pi^{dep.goal}(G)$ contains the following rule:

$$finally(neg(g)).$$

For instance:

$finally(on(a, c)).$

$finally(on(c, b)).$

$finally(ontable(d))$.

2. *The domain independent part* $\pi^{indep}(D, O, G, l)$: This part is independent from the content of particular domain.

   (a) *Defining time*: In answer set planning, we need to give either the exact bound or at least an upper bound of the plan lengths that we want to consider. This is what makes each answer set finite. The encoding π^{indep} depends on a constant referred to as *length* which is the upper bound of the length of plans that we intend to consider. Using this *length* we define a predicate *time* which specifies the times points of our interest.

   $time(1). \ldots time(length)$.

   (b) *Defining goal*(T): The following rules define when all the goal conditions are satisfied.

   $not\_goal(T) \leftarrow time(T), finally(X), \neg holds(X, T)$.

   $goal(T) \leftarrow time(T), \neg not\_goal(T)$.

   (c) *Eliminating possible answer sets which do not have a plan of the given length*: The following constraint eliminates possible answer sets where the goal is not satisfied in the last time point of interest.

   $\leftarrow \neg\ goal(length)$.

   (d) *Defining contrary*: The following facts define when a fluent literal is the negation of the other.

   $contrary(F, neg(F))$.

   $contrary(neg(F), F)$.

   (e) *Defining executability*: The following two rules use the executability conditions to define when an action A is executable in a time T. Note that we are only interested in times before the time point denoted by *length*, as no actions are

supposed to occur after that.

$not\_executable(A,T) \leftarrow exec(A,F), \neg holds(F,T).$

$executable(A,T) \leftarrow T < length, \neg not\_executable(A,T).$

(f) Fluent values at time point 1:

$holds(F,1) \leftarrow initially(F).$

(g) *Effect axiom*: The following rule describes the change in fluent values due to the execution of an action.

$holds(F,T+1) \leftarrow T < length, executable(A,T), occurs(A,T), causes(A,F).$

(h) *Inertia*: The following rule describes which fluents do not change there values after an action is executed. In the literature, this is referred to as the frame axiom.

$holds(F,T+1) \leftarrow$

$$contrary(F,G), T < length, holds(F,T), \neg holds(G,T+1).$$

(i) *Occurrences of actions*: The following rules enumerate action occurrences. They encode that in each answer set at each time point only one of the executable actions occurred. Also, for each action that is executable in an answer set at a time point, there is an answer set where this action occurs at that time point.

$occurs(A,T) \leftarrow action(A), time(T), \neg goal(T), \neg not\_occurs(A,T).$

$not\_occurs(A,T) \leftarrow action(A), action(AA), time(T), occurs(AA,T), A \neq AA.$

$\leftarrow action(A), time(T), occurs(A,T), \neg executable(A,T).$

5.2.3 Language $\mathcal{B}$

Language $\mathcal{B}$ results when Language $\mathcal{A}$ is extended to allow *static causal propositions*. Using *static causal propositions* we can express knowledge about the states directly in

terms of what are possible or valid states and this knowledge can be used to indirectly infer effects of actions, executability of actions, or both. A static causal proposition is of the form

$$p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r \textbf{ causes } f \tag{5.5}$$

where $p_1, \ldots, p_n, q_1, \ldots, q_r$ are fluents and f is either fluent literal, or a special symbol $false$.

We say a state σ satisfies a static causal proposition of the form 5.5 if: (i) f is a literal and $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ hold in σ implies that f holds in σ; or (ii) f is false and at least one of $p_1, \ldots, p_n, \sim q_1, \ldots, \sim q_r$ does not hold in σ. Two examples of static causal propositions are $position(X)$ **causes** $\sim position(Y)$ **if** $X \neq Y$ and $married\_to(X) \wedge married\_to(Y) \wedge X \neq Y$ **causes** $false$. In particular, the static causal proposition in the second example expresses a constraint since the fluent corresponds to $false$.

Given a domain description D consisting of effect propositions, executability conditions, and static causal propositions, and a set of observations O about the initial situation we say σ_0 is an initial state corresponding to D and O if (i) for all the observations of the form **initially** f in O, f holds in σ_0 , and (ii) σ_0 satisfies the static causal propositions in D.

We say $\sigma_0, a_1, \sigma_1, a_2, \sigma_2 \ldots a_n, \sigma_n$ is a valid **trajectory** of (D, O), if (a) σ_0 is an initial state corresponding to (D, O), (b) for $1 \leq i \leq n$, a_i is executable in σ_{i-1}, and (c) for $1 \leq i \leq n$, $\sigma_i \in \Phi_D(a_i, \sigma_{i-1})$.

More details about static causal propositions, answer set encoding od of planning problems expressed in Language $\mathcal{B}$ can be found in [4].

5.3 A language for Planning Preferences: $\mathcal{PP}$ Language

Language $\mathcal{PP}$ [43] is a language for the specification of user preferences with a logic programming implementation of it, based on answer set programming. Language $\mathcal{PP}$ allows someone to specify preferences among feasible plans and temporal preferences over plans too. The preferences representing time are expressed using the temporal connectives *next, always, until* and *eventually*. There are three different classes of preferences in $\mathcal{PP}$ [43] *basic desires*, *atomic preferences* and *general preferences*. In this work we studied the two first classes of preferences because seems that they are useful to express preferences in evacuation plans.

A *basic desire*, denoted as φ, is a $\mathcal{PP}$ formula expressing a preference about a trajectory with respect to the execution of some specific action or with respect to the states that the trajectory gets when an action is executed. In particular, $\mathcal{PP}$ defines *goal preferences* to define preferences in the final state.

The set of basic desires of language $\mathcal{PP}$ can be defined inductively by the following context-free grammar $G_{PP} := (N, \Sigma, P, S)$, such that $N := \{S\}$ is the finite set of non terminals; $\Sigma := \mathbf{A} \cup \mathcal{F}_F$ is the finite set of terminals ($N \cap \Sigma = \emptyset$) where $\mathbf{A}$ and $\mathcal{F}_F$ represent the set of actions of the problem and the set of all fluent formulas (propositional formulas based on fluent literals) respectively; $S \in N$ is the initial symbol of the grammar; and $P := \{S \rightarrow p|\mathbf{goal}(p)|occ(a)|S \wedge S|S \vee S|\neg S|\mathbf{next}(S)|\mathbf{until}(S,S)|\mathbf{always}(S)|\mathbf{eventually}(S)\}$ is the finite set of productions or rules where $p \in \mathcal{F}_F$ and $a \in \mathbf{A}$.

The following definition from [43] allows one to check whether a trajectory satisfies a basic desire formula where, given a trajectory $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n$ the notation $\alpha[i]$ denotes the trajectory $s_i a_{i+1} s_{i+1} \ldots a_n s_n$.

Definition 5.1. [43] Let $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots a_n s_n$ be a trajectory, and let φ be a basic desire formula. α satisfies φ (written as $\alpha \models \varphi$) iff

- $\varphi = \textbf{goal}(\psi)$ and $s_n \models \psi$
- $\varphi = \psi \in \mathcal{F}_F$ and $s_0 \models \psi$
- $\varphi = occ(a)$, $a_1 = a$, and $n \geq 1$
- $\varphi = \psi_1 \wedge \psi_2$, $\alpha \models \psi_1$ and $\alpha \models \psi_2$
- $\varphi = \psi_1 \vee \psi_2$, $\alpha \models \psi_1$ or $\alpha \models \psi_2$
- $\varphi = \neg\psi$ and $\alpha \not\models \psi$
- $\varphi = \textbf{next}(\psi)$, $\alpha[1] \models \psi$, and $n \geq 1$
- $\varphi = \textbf{always}(\psi)$ and $\forall(0 \leq i \leq n)$ we have that $\alpha[i] \models \psi$
- $\varphi = \textbf{eventually}(\psi)$ and $\exists(0 \leq i \leq n)$ such that $\alpha[i] \models \psi$
- $\varphi = \textbf{until}(\psi_1, \psi_2)$ and $\exists(0 \leq i \leq n)$ such that $\forall(0 \leq j < i)$ we have that $\alpha[j] \models \psi_1$ and $\alpha[i] \models \psi_2$.

□

Definition 5.1 will also allow one to compare trajectories.

Definition 5.2. [43] Let φ be a basic desire formula and let α and β be two trajectories. The trajectory α is preferred to the trajectory β (denoted as $\alpha \prec_\varphi \beta$) if $\alpha \models \varphi$ and $\beta \not\models \varphi$.

We say that α and β are indistinguishable (denoted as $\alpha \approx_\varphi \beta$) if one of the two following cases occur: (i) $\alpha \models \varphi$ and $\beta \models \varphi$, or (ii) $\alpha \not\models \varphi$ and $\beta \not\models \varphi$. □

Whenever it is clear from the context, we will omit φ from $\prec_\varphi$ and $\approx_\varphi$.

Definition 5.3. [43] Let φ be a basic desire formula. A trajectory α is said to be a most preferred trajectory w.r.t. φ, if there is no trajectory β such that $\beta \prec_\varphi \alpha$. □

Example 5.2. [43] Let us considering example given in Subsection 5.2.2 about the blocks world domain where there are four blocks $a,\ b,\ c,\ d$ on a table and the goal is to have block a on block c and block c on block b. We also presented the three feasible plans p_1, p_2 and p_3:

1. $p_1 = unstack(a, b)\ put\_down(a)\ unstack(d, c)\ put\_down(d)\ pick\_up(c)\ stack(c, b)$ $pick\_up(a)\ stack(a, c)$.
2. $p_2 = unstack(d, c)\ put\_down(d)\ unstack(a, b)\ put\_down(a)\ pick\_up(c)\ stack(c, b)$ $pick\_up(a)\ stack(a, c)$.
3. $p_3 = unstack(d, c)\ put\_down(d)\ unstack(a, b)\ stack(a, d)\ pick\_up(c)\ stack(c, b)$ $unstack(a, d)\ stack(a, c)$.

To express that we would like to stack a on top of d and unstack a from the top of d, we can write the following basic desire:

$\varphi_1 = \mathbf{eventually}(occ(stack(a, d) \wedge occ(stack(a, d))$

If we do not desire to have a on the table, we can write the basic desire:

$\varphi_2 = \mathbf{always}(\neg ontable(a))$.

Moreover if α_1, α_2 and α_3 are the trajectories corresponding to plans p_1, p_2 and p_3 respectively then $\alpha_3 \prec_{\varphi_1} \alpha_2$ (α_3 is preferred to the trajectory α_2) and $\alpha_3 \prec_{\varphi_1} \alpha_2$ (α_3 is preferred to the trajectory α_2). □

An *atomic preference* represents an ordering between basic desire formulas. It indicates that trajectories that satisfy φ_1 are preferable to those that satisfy φ_2, etc. Clearly, basic desire formulas are special cases of atomic preferences.

Definition 5.4. [43] An atomic preference formula is defined as a formula of the type $\varphi_1 \lhd \varphi_2 \lhd \ldots \lhd \varphi_n$ $(n \geq 1)$ where $\varphi_1, \varphi_2, \ldots, \varphi_n$ are basic desire formulas. □

Then the definition to compare trajectories w.r.t. atomic preferences is the following:

Definition 5.5. [43] Let α, β be two trajectories, and let $\Psi = \varphi_1 \lhd \varphi_2 \lhd \ldots \lhd \varphi_n$ be an atomic preference formula.

— α, β are indistinguishable w.r.t. Ψ (written as $\alpha \approx_\Psi \beta$) if $\forall i.[1 \leq i \leq n \Rightarrow \alpha \approx_{\varphi_i} \beta]$.

— α is preferred to β w.r.t. Ψ (written as $\alpha \prec_\Psi \beta$) if $\exists(1 \leq i \leq n)$ such that **(a)** $\forall(1 \leq j < i)$ we have that $\alpha \approx_{\varphi_j} \beta$, and **(b)** $\alpha \prec_{\varphi_i} \beta$. We will say that $\alpha \preceq_\Psi \beta$ if either $\alpha \prec_\Psi \beta$ or $\alpha \approx_\Psi \beta$. □

5.3.1 Computing answer sets of planning problems with preferences

In order to compute the preferred trajectories of a planning problem expressed in an action language w.r.t. ψ a preference of any of the three classes, in [43] is defined the answer set encoding $\Pi(D, O, G, l, \psi)$ as $\pi(D, O, G, l) \cup \Pi_\psi \cup \Pi_{sat}$ where $\pi(D, O, G, l)$ is the answer set encoding of the planning problem as defined in [4], Π_ψ is the encoding of the preference formula ψ and Π_{sat} are the set of rules for checking of basic desire formula satisfaction. Moreover, if M is an answer set of $\pi(D, O, G, l)$, then α_M denotes the plan achieving the goal G represented by M.

It is worth mentioning that in particular [43] shows how we can obtain the most preferred trajectory with respect to a basic desire or an atomic preference. It is assigned a weight to each component of the preference formula, then the weight of each trajectory is obtained based on the weight of each component of the preference formula satisfied by the trajectory. Finally, in order to obtain the most preferred trajectory, i.e., the answer sets with maximal weight it is used the **maximize** construct in SMODELS. In [43] it is recommended to use **jsmodels** since SMODELS has some restrictions on using the **maximize** construct. Moreover, in [43] it is showed how an atomic preference of $\mathcal{PP}$ can be mapped to a collection of standard ordered rules as defined by Brewka in order

to obtain the most preferred trajectory. However, the use of weights or the mapping results in a complicated encoding. In Subsection 8.3.1 we show that extended ordered rules with negated negative literals allows a simpler and easier encoding.

5.4 Conclusion

In this chapter we presented an overview about the Logic-based Planning approaches, in particular we presented an overview about one of the AI planning approaches to model and solve planning problems in a natural and elegant way is Answer Set Planning.

In the contribution part of this work we shall use Answer Set Planning to specify and encode the evacuation planning problems.

In this chapter we also studied Language $\mathcal{PP}$ [43] that allows one to specify preferences among feasible plans and temporal preferences over plans too. Moreover in Contribution Part of this work we shall present an extension for Language $\mathcal{PP}$ and we shall show how this language and its extension can be applied to obtain the preferred evacuation plans.