Integrating Conditional Planning and Robust Execution in HRI

Candidate:

Valerio Sanelli

Thesis Advisor:

Luca Iocchi

SAPIENZA
UNIVERSITÀ DI ROMA

Chapter 1

Introduction

1.1 Planning

1.1.1 Planners Characteristics/Requirements

1. How to handle failures → Replanning avoiding loops

2. How to represent uncertainty → Probabilistic reasoning

3. How to deal with Exogenous Events → Implicit/Explicit Modelling

4. Hyerarchical Tasks

5. Frame Problem: how to represent the fact that things do not arbitrarly changes.

6. Qualification problem: how to deal with the things that prevent me from achieving my intendend result (impossibility of list alla preconditions).

7. How to model/deal time (explicit/implicit)

8. World-based (STRIPS) vs Plan-based Planner (Partial-order Planning)

9. Total-order (HTN) vs Partial-order planning

10. Variables Observation, Exception generation and handling

11. Evaluate solution quality, planning time and the ability to avoid dead-end states

12. Exploit the restrictions placed on the states and actions, on observability, and on the value function and optimality criterion

13. Computational complexity of the plan existence problem

14. Tradeoff between complexity of representation and execution

1.1.2 Partial Ordering

Partial-order planning, as opposed to total-ordering, is a search in plan space where every node of the problem represents a partial plan. A plan is complete iff every precondition is achieved and that happen when the precondition is the effect of an earlier step and no possibly intervening step undoes it. A plan is represented as

- a set of steps

- ordering constraints

- causal links from outcome of one step to precondition of another

- open preconditions

A start step has the initial state description as its effect while a finish step has the goal description as its precondition. The search procedure picks one open precondition of an action B, and one action of A that achieves the precondition of B. Add the causal links for the actions and the ordering constraints for A,B, *Start*, *Finish* and tries to resolve conflicts when possible, otherwise backtracks. A conflict is resolved by adding ordering constraints called demotion and promotion that move the conflictual action after of before the causal link, respectively. When there are no more open preconditions the goal is satisfied. Partial-order planning produces more trivial serializability than total-order planning that allows performing quickly when dealing with

goals that contain sub-goals. A sub-goal is trivially or laboriously serializable depending on the search space. For the stated properties,Partial-order planning is more likely to find the quickest path. Not only, it could be very difficult to write good knowledge bases for Total-order planner because the total-ordering requirement for the subtasks is restrictive. One drawback of this type of planning system is that it requires a lot more computational power for each node. This depends on the higher complexity of this type of algorithms and has a huge impact on the applicability on robots in real world scenarios because the energy consumption.

1.1.3 Uncertainty

Uncertainties generally interfere with two aspects of planning:

- **Predictability:** due to uncertainties it is not known what will happen in the future when certain actions are applied. This means that future states are not necessarily predictable.

- **Sensing:** due to uncertainties, the current state is not necessarily known. Information regarding the state is obtained from initial conditions, sensors, and the memory of previously applied actions.

Classical planning is fast, but cannot reason about multiple action outcomes, unreliable observations, or multiple possible worlds. Decision-theoretic planning, by contrast, produces a policy maximizing the probability of success (expected value) from any belief state the robot might reach during plan execution, taking account of the probabilities of every possible action outcome in those states. Decision-theoretic planning handles multiple action outcomes (Markov decision processes, MDPs) and unreliable observations (POMDPs), but at an infeasible computational cost given the size of typical robot domains.

1.1.4 Epistemic

1.1.5 Exogenous Event

A robotic system must deal with the external world that is, by its own nature, a dynamic system itself. This external world produce a series of stimuli that must be treated carefully when accomplish a task. The interaction between the two systems is two-way, since the robot could influence the state of the world through its action and vice versa. The actions performed by the robot are called *control* actions, while the actions that the world exert on the robot are called *exogenous* actions. Is it possible to model this events in two ways: implicit-event model and explicit-event model. In implicit-event models the effects of the exogenous event are folded into the transition probabilities associated with the action. Instead, when the effects of actions are decomposed so that a transition is determined by the effects of agent actions and external events that occur with a certain probability we call it explicit-event model.

1.1.6 qualification problem, ramification problem, frame problem

PDDL

- Frame problem: everything not explicitly change by effects persist.

- Qualification problem: as SitCalc,can never finish listing all the required preconditions and possible conditional outcomes of actions

Chapter 2

Related Works

2.1 Human Robot Interaction

Social Rules Norms of human being imitated/reproduced by the robot in order to encourage collaboration and that are socially acceptable.

Cooperation doesn't mean necessarily social, and this is the case of classic automation approach. The most promimenent cases are the ones in which the human is encouraged to collaborate by the means of a reward (griedy case). Human are also curious, so encourage this part could also help in establish and mantain a collaboration through time. (curiosity case) The third and most interesting case for this study is the scenario in which the robot itself starts the collaboration. There are many aspects to take into account in this, and I will enlist the most important in the following. A robot must be capable of discerning cases in which an interaction is desirable and cases in which is not. Like in the simple case of a "welcome robot", where is important that the robot says "hi" when a person shows in front of him but it must also take into account the fact that a person could appear in front of him multiple times. Another important but maybe trickier case is to equip the robot with the capability of understanding the mood of the person and than react consequently. Although this could be seen as a fictional case, since is impossible at the moment even for the most complex cognitive robot

to understand emotions, there are some simple cases that could be treated. The goal, in this case, is not to reproduce a perfect model of the human mind state and how it could react to some sorts of events, but rather discover some patterns that could lead to undesirable outcomes. For instance, a person could be busy with its work and the interference of the robot, when is not helpful in the context of the human task, could be perceived as noise. Thus is important to maintain a representation of the person daily routines and enrich this when is possible with the information that comes from the social interactions that occurs from time to time between the two. If we are interested in developing robots strictly social, we cannot leave aside the emotional aspect of the interaction.

Main Questions:

1. How to start/mantain interaction

2. How to represent parallel/serial interaction

3. How to represent/update common knowledge

2.1.1   Industrial Cases (No HRI)

Those are the cases in which robot and human are both aware of the plan and they cooperate deterministically in order to achieve the goal. There is no HRI as the human is seen as another agent in the environment.

- Assisted assembly

- Logistic

- Space

2.1.2   Almost Social Cases (HRI Not Mandatory)

In this cases the interaction between the human and the robot is minimal. The human is a target and the interaction is a secondary objective. No representation of the human state is requred.

- Patrolling

- Search & Rescue

2.1.3   Strictly Social Cases (HRI Mandatory)

Those cases represents the most significant for this work. In this case the interaction is the principal objective and the robot itself could start the interaction. A representation of the human state is mantained.

- COACHES

- Assistance task

- Waitress Robot

Chapter 3

Plan Representation

3.1 Formalisms

3.1.1 STRIPS (Classical Planning)

STRIPS[3] stand for STanford Research Institute Problem Solver.

It is a formal language (and a planner) used to describe deterministic, static, fully observable domains.

In STRIPS the initial state is assumed to be known with certainty and any action taken at any state has at most one successor (no multiple outcomes). It is not possible to describe indirect effects of actions, as action preconditions/postconditions are expressed simply as conjunction of literals (positive for preconditions).

More formally, a STRIPS instance is a quadruple $< P, O, I, G >$, where:

- P is a set of conditions (propositional variables).

- O is a set of operators (actions). Each action is itself a quadruple with four set of conditions specifying which conditions must be true/false for the action to execute and which conditions are made true/false after the execution

- I is the initial state, specified by a set of conditions that are initially true (all others are false).

- G specify the goal conditions. This is a pair of two set specifying which conditions must hold true/false in a state in order to be considered a goal.

A plan computed by STRIPS is a linear sequence of actions from initial state to goal state with no branches (i.e. no conditional effects).

This makes STRIPS very fast: the search is done off-line and then the found plan is executed on-line with "eyes closed".

On other hand it cannot deal with uncertainty in both sensing/knowledge, like unreliable observations, incomplete/incorrect information or multiple possible worlds.

An expanded version of STRIPS is Action Description Language (ADL)[5], which expand STRIPS with some syntactics features:

- state representation: allows also negative literals.

- action specification: add conditional effects and logical operators for express them.

- goal specification: quantified sentences and disjunctions.

3.1.2   PDDL

PDDL[6] is the standard de facto for planning domains description. It generalizes both STRIPS[3] and Action Description Language (ADL [5]).

It is intended to describe the "physics" of a domain in terms of predicates, possible actions and their effects.

States are represented as a set of predicates with grounded variables and arbitrary function-free first-order logical sentences are allowed to describe goals. Action effects are the only source of change for the state of the world and could be universally quantified and conditional. This makes PDDL asymmetric: action preconditions are considerably more expressive than action effects.

The standard version of PDDL divide the planning problem in two parts: domain description and problem description. Domain description contains

all the element that are common to every problem of the problem domain while the problem description contains only the elements that are specific for that instance of problem. This two constitute the input for the planner. The output is not specified by PDDL but it is usually a totally/partially ordered plan (a sequence of actions that may be also executed in parallel). The domain description consist of:

- **domain-name**.

- **requirements**. declares to planner which features are used. some examples are strips, universal-preconditions, existential-preconditions, disjunctive-preconditions, adl. When a requirement is not specified, the planner skips over the definition of that domain and won't cope with its syntax.

- **object-type hierarchy**.

- **constant objects**.

- **predicates**.

- **actions**. are operators-schemas with parameters, that will be grounded/instantiated during execution. They had parameters (variables to be instantiated), preconditions and effects. Effects could be also conditional.

The problem description is composed by:

- **problem-name**.

- **domain-name**. to which domain the problem is related.

- **objects**. the definition of all possible objects (atoms).

- **initial conditions**. conjunction of true/false facts, initial state of the planning environment.

- **goal-states**. a logical expression over facts that must be true/false in a goal-state.

I refer to [6] for a more formal definition of the syntax. Through the years PDDL has received several updates (the last version is PDDL 3.1) and extensions. The most notable are:

- PPDDL[15] is a probabilistic version of PDDL.
  It extends PDDL 2.1 with probabilistic effects, reward fluents, goal rewards and goal-achieved fluents. This allows realising Markov Decision Process (MDP) planning in a fully observable environment with uncertainty in state-transitions

- RDDL[16] introduce the concept of partial observability in PDDL. It allows to describe (PO)MDP representing states,actions and observations with variables. Semantically, RDDL represent a Dynamic Bayes Net (DBN) with many intermediate layers and extended with a simple influence diagram utility node reresenting immediate reward.

- KPDDL[17] add to PDDL some constructs to represent the incomplete knowledge of the agent about the environment.
  It is discussed in next section.

3.1.3 $ALCK_{NF}$

It is an autoepistemic logic, an extension of ALC with the add of the epistemic operator **K** and default assumption operator **A**. It is presented in [14]. The logic ontology assumes the existence of *concepts* and *roles*.
A concept represent a class of *individuals* while the roles model the relations between classes.
The interpretation structure of Default logics is a Kripke structure, with labelled nodes and edges. Each node represents an individual and is labelled with the concept corresponding to the individual while each edge is a role.
This structures could be interpreted as a dynamic system where the individuals are the states of the system that holds the properties valid in that state (like fluents in situation calculus). Edges represent transactions between system states and are labelled with the action that causes the change in state.
So each node/individual represent an *epistemic state* of the robot so that

there is an edge between two nodes if the associated action modifies the source epistemic state in the destination epistemic state.

There are two types of actions: ordinary actions and sensing actions.

Ordinary actions are deterministic actions with effects that depend on the context they are applied, while the sensing actions do not modify the environment but the knowledge of the agent since they evaluate boolean properties of the environment.

Thanks to the latter it is possible to explicitly model the effects that this type of actions has on the knowledge and to characterise plans that the agent could actually execute.

It is also possible to specify actions that can run in parallel. Parallel actions in a state can be executed iff each could be executed individually in that state and the effects of the actions are mutually consistent.

For what concerns the Frame problem, the language use default axioms to represent the persistence of properties and frame axioms that express the causal persistence of rules.

Epistemic operators guarantee that the generated plan is actually executable, unlike first-order logic, and sensing actions produces ramifications that could help to reach the goal. This type of plan, with sensing plus ordinary actions, is called conditional plan.

3.1.4 KPDDL

KPDDL[17] is an extension of PDDL based on the logic $ALCK_{NF}$.

Its aim is to augment the planning domain with the (incomplete) knowledge of the agent about the environment.

KPDDL introduces several specifications:

- *sense terms*. effects of actions that helps to disambiguate the true state of the environment.

- *non-inertial terms*. specifies non-inertial properties.

- *strongly-persist terms*. specifies strongly persistence properties.

- *formula-init term.* is the specification for an incomplete initial state.

The fundamental contribute of KPDDL from the semantic viewpoint is the interpretation of a dynamical system through its epistemic state. An epistemic state represents what the agent knows about the world, that could be different from what is true in the world, and the planning is realised by modelling at this meta-level.

A planning problem can be specified as:

$$KB \cup \phi(init) \models \pi_\psi(init)$$

where KB is the set of axioms representing the planning domain, $\phi(init)$ is the assertion representing the initial state and $\pi_\psi(init)$ is an $ALCK_{NF}$ assertion that is true in the initial state iff executing the plan agent knows ψ. The associated planner produces conditional plans in the form of conditional trees (or direct acyclic graphs), in which the root is the initial states, the leaves are goal states and the branches are different results of sensing. Conditional plans are distinguished in *weak/strong*, whether or not are present leaves that are not goal states and from which a goal state could not be reached. Plans could be also *cyclic*, in which case the termination is not guaranteed. Formally, a plan is defined as quadruple $P = <G(V,E), I, F_G, F_F>$ where:

- G is a direct graph, with V nodes and E edges.

- $I \in V$ is the initial state.

- $F_G \subseteq V$ is the set of final states in which the goal is achieved.

- $F_F \subseteq V$ is the set of final states in which the goal is not achieved.

3.1.5   Answer Set Programming (ASP)

Answer set programming (ASP) is a form of declarative programming oriented towards difficult, primarily NP-hard, search problems. [9] [10]
It's declarative in the sense that it describe the problem, not how to solve it. In particular, a problem instance I is translated into a non-monotonic logic program P and passed to an ASP solver which will search the stable models

13

of P. From these, the solution of I is extracted.

ASP programs are composed of Prolog-like rules, but it uses a different computational system. In fact, ASP solvers are based on DPLL algorithm, that in general guarantees the termination (unlike SLDNF resolution in Prolog).

A rule has the form:

```
<head> :- <body>
```

where $<head>$ is a disjunction of positive/negative literals, while $<tail>$ is a conjunction of positive/negative literals.

The symbol :- is dropped if $<body>$ is empty; this are called *facts*. Rules with an empty $<head>$ are called *constraints* and are used to reduce the number of possible solutions (stable models). These are useful to express action preconditions like in other formalism (for instance 3.1.2).

ASP includes two form of negation: strong ("classical") negation and negations as failure. This allow to represent defaults, useful in order to express the closed world assumption and characterize a solution to the frame problem (inertia of the world: *"everything is presumed to remain in the state in which it is"*).

I will now introduce some examples from [9] to illustrate ASP properties:

```
(1) p :- q.
(2) cross :- not train.
(3) cross :- -train.
```

(1) is an example of a classic Prolog like rule. It can be interpreted as $q \to p$, that is *"if **q** hold, than **p** is in the stable model"*.

(2) represent an example of negation as failure. This rule represent the knowledge: *"it is safe cross if there are no information about an approaching train"* (not a good idea indeed).

(3) is preferable in this case, since uses the strong negation: *"it is safe cross if it is known that the train is not approaching"*.

If we combine both negations it is possible to express the closed world assumption (*"a predicate does not hold unless there is evidence that he does"*):

14

```
-q(X,Y) :- not q(X,Y), p(X), q(Y).
```

interpreted as *"q does not hold for a pair of element of p if there are no evidence that it does"*. This allow to include also negative facts about q, so to specify the closed world assumption for some predicates and leave the others with the open world assumption.

Finally, it is possible to express time explicitly with constants. This feature is used to define a metric for plans and for express the *"frame default"*:

```
p(T+1) :- p(T), not -p(T+1), time(T).
```

"p will remain true at time T+1 if there is no evidence that becomes false". ASP programs are often organized in three sections: generate, test and define. The *"generate"* section describes the set of possible solutions while the *"test"* section is used to discard bad solutions. The *"define"* section defines auxiliary predicates used in the constraints. The order of the rules in an ASP program doesn't matter.

ASP has been widely used in the service robotics context, both in simulated or real scenarios. Some examples are [11] and [12].

However, ASP is not well suited for modelling/reasoning about uncertainty in a domain. Incomplete information cannot be handled correctly due to the fact that stable model semantics is restricted to boolean values. Not only, there is no notion of probability and thus all stable models are equally probable. Some extension to ASP with probabilistic information has been proposed, like in [13] where Markov Logic Networks (MLN) are used.

3.1.6   Situation Calculus

Situation Calculus[7][8] is a logic formalism used to represent and reason about dynamical systems. It allows to express qualitative uncertainty about the initial situation of the world and the effects of actions through disjunctive knowledge.

The state is represented as a set of first-order logic formulae. The basic elements are actions, fluents and situations.

A *Situation* is a first order term denoting a state and is characterised by the sequence of actions applied to the initial state in order to reach it.

Fluents are functions or predicates that have as last argument the situation of application and represents properties of a particular situation. Actions may also be parametrized with variables. A fluent differs from a (normal) predicate or function symbol as its value may change from situation to situation.

The binary function symbol $do(\alpha, s) \rightarrow s'$ represent the result of apply action α on a situation s, where s' is the resulting situation.

The binary predicate symbol $Poss(.,.)$ represents the possibility of executing an action. For instance, $Poss(\alpha, s)$ is true iff is possible to apply α in situation s.

A Planning Problem in the Situation Calculus is then defined as

$$\mathcal{D} \vDash \exists s. Goal(s)$$

where $\mathcal{D}$ is a theory of actions, which consists of:

- **Axioms for the Initial Situation**. A formula that represent properties verified in the initial situation S_0.

- **Unique Name Axioms for the Actions**.

$$do(\alpha_1, s_1) = do(\alpha_2, s_2) \Rightarrow \alpha_1 = \alpha_2 \wedge s_1 = s_2$$

  Unique name for actions and situations.

- **Precondition Axioms**. They codify necessary properties in order to execute actions.

- **Successor State Axioms**. For each fluent establish if is true after the execution of an action. Those axioms solve the frame problem as they enforce the persistence of those properties that are not involved or changed by the current actions, describing the causal laws of the domain with an axiom per fluent.

A plan is a sequence of actions that brings the system from the initial situation to a situation that satisfies the condition *Goal(s)*, whenever the formula representing the goal is satisfiable.

3.1.7 Markov Decision Process (MDP)

A Markov Decision Process is a stochastic machine state representation of a dynamic system, in which every transition has assigned a probability. This process is 'Markovian' since Markov property hold, that is the future state depends only on the current action and state and not from the past states.
The actions to take in every state are specified by a utility function, that guide the agent through the goal and represent the utility of being in a state or take an action. This function specifies a preference over the paths of the graph.
Actions are the only source of change and are assumed to be instantaneous. In MDP there is no explicit modelling of time. Nor the world dynamics or the reward function depend on absolute time.
A policy $\pi : S \to A$ is a total function that specifies for each state the action to take. The planning problem is then an optimisation problem in which the objective is to find the policy that maximises the expected utility associated with the states and the action. The goal is represented as a reward and is a deterministic function of the current state.
This it is a quantitative representation, unlike the previously introduced formalisms, in which the uncertainty is represented through probabilistic means (see also 3.1.8).

More formally, an MDP is a 5-tuple $\sum =< S, A, P(.,.), R(.,.), \gamma >$ where:

- S is a finite set of states

- A is a finite set of actions

- $P_a(s'|s)$ is the probability that action $a \in A$ in state $s \in S$ at time t will lead to state $s' \in S$ at time $t+1$. Notice that $\sum_{s' \in S} P(s, a, s') = 1$.

- $R_a(s'|s)$ is the immediate reward received after the transition from s to s'.

- $\gamma \in [0,1]$ is the discount factor, which weights the importance of the future versus the present rewards.

The probability of a sequence of states $< s_0, s_1, ..., s_n >$ is called history and is defined given a policy as:

$$P(h|\pi) = \prod_{i \geqslant 0} P_{\pi(s_i)}(s_{i+1}|s_i)$$

The utility function is specified by associating for each $< state, action >$ a cost/reward depending on the value that transition must have for the agent. If we define a cost function as $C : S \times A \rightarrow \mathbb{R}$ and a reward function as $R : S \rightarrow \mathbb{R}$, is it possible to define the utility of execute an action a in the state s as:

$$V(s|a) = R(s) - C(s,a)$$

Consequently the utility of a policy π in a state s is defined as:

$$V(s|\pi) = R(s) - C(s, \pi(s))$$

and thus for the history:

$$V(h|\pi) = \sum_{i \geqslant 0} R(s_i) - C(s_i, \pi(s_i))$$

This sum usually doesn't converge and we need a discount factor γ in order to reduce the effect of costs/rewards distant from the actual state. Introducing γ, we can rewrite the utility of the history as:

$$V(h|\pi) = \sum_{i \geqslant 0} \gamma_i (R(s_i) - C(s_i, \pi(s_i)))$$

with $0 < \gamma < 1$. With a value of γ near 1 we promote immediate rewards over futures, while with $\gamma \cong 0$ the vice versa. From the history utility is possible to calculate the expected value of a policy utility by considering all the history that the policy induce. Given H as the et of all possible history induced by π, this expected value is:

$$E(\pi) = \sum_{h \in H} P(h|\pi)V(h|\pi)$$

a policy π^* is said to be optimal for a stochastic system Σ if:

$$E(\pi^*) \geqslant E(\pi) \quad \forall \pi$$

where π is an admissible policy for Σ. A solution to an MDP is then a policy that, once found, is applied in any state in a deterministic fashion.

3.1.8 Partially Observable MDP (POMDP)

In many cases, the assumption of complete observability for the state is too strict so Partially Observable MDP was introduced.

In this case, a set of observations is defined that characterise the observable part of the stochastic system Σ.

A POMDP is a stochastic system $\Sigma =< S, A, P(.,.) >$ as defined in 3.1.7, with the addition a finite set of observations O with probability $P_a(o|s)$ for all $a \in A, s \in S, o \in O$. $P_a(o|s)$ represent the probability of observe o in the state s after the execution of a. This is defined for every state/action and the condition $\sum_{o \in O} P(o|s) = 1$ hold.

More than one observations could correspond to one state (i.e. $P_a(o|s) = P_a(o|s')$) and thus is not possible to obtain the actual state from the observations. Analogously the same state could correspond to different observations depending on the action executed, that is $P'_a(o|s) = P_a(o|s)$. Thus observations depend both on the state and on the action. For instance, a sensing action does not modify the state of the system but could lead to a different observation.

Probability distributions over the states are called *belief states*. Let be b one belief state and B the set of all belief states. Then $b(s)$ denotes the probability for the system to be in state s.

In order for $b(s)$ to be a probability distributions both $0 \leqslant b(s) \leqslant 1$ and $\sum_{s \in S} b(s) = 1 \ \forall b \in B$ must hold. Given an action a and a belief state b, the next belief state deriving from apply a in b is:

$$b_a(s) = \sum_{s' \in S} P_a(s|s')b(s')$$

Similarly we could compute the probability of observe o given the action a:

$$b_a(o) = \sum_{s \in S} P_a(o|s)b(s)$$

At last, we can compute the probability of being in s after executing a and observing o:

$$b_{a,o}(s) = \frac{P_a(o|s)b_s(s)}{b_a(o)}$$

Planning in POMDPs is formulated as an optimization problem over the belief space to determine the optimal policy $\pi^* : B \to A$ with respect to an utility function defined as in 3.1.7. In fact we can see at a POMDP as a continuous MDP where the continuous space is the belief space.

The expected value for the utility of a belief state b is defined as:

$$E(b) = \min_{a \in A} C(b,a) + \gamma \sum_{o \in O} b_a(o)E(b_{a,o})$$

where:

$$C(b,a) = \sum_{s \in S} C(s,a)b(s)$$

3.1.9 Analysis of Formalisms

The introduced formalisms and models show the essential bond between expressiveness and computational complexity. The planning problem in classic logic is decidable but the complexity varies from constant to EXPTIME depending on which characteristics are added to the language.

For instance, STRIPS is the less expressive of the presented formalisms, still deciding whether a plan exists is PSPACE-complete [4]. Many restrictions can be enforced in order to make this problem decidable in polynomial time, like restricting the type of formulas or the number of pre-postconditions.

$ALCK_{NF}$ is an example of decidable formalism with a good expressiveness, since it allows to express the agent epistemic state explicitly. However

the planning in its most general case falls in PSPACE complexity, thus intractable for an online approach. In this and similar cases, partial planning and restricted version of the formalism are used in order to solve the complexity problem.

In KPDDL is it possible to express incomplete information through epistemic states and define sensing actions that help to disambiguate the states. In practice, however, it was seen that even a small variation in the number of states makes the problem intractable, proving that the class of complexity is at least PSPACE (instance checking/reasoning in $ALCK_{NF}$ is PSPACE-complete [14]).

Talking about POMDPs models, they are often computationally intractable to solve exactly and thus it is necessary to introduce methods to approximate solutions. Some examples are [18] and [19], where particle filters and PCA are respectively used to represent the belief state compactly or exploit its properties.

3.2   Bechmarks Experiments

In the following I will enlist some benchmarks domain problems.
Most are common to the planning community and are taken from international competitions like IPC (International Planning Competition).
Others are obtained reasoning about problems with particular structures or properties useful to test the chosen planners.
It is somehow difficult to derive a complete and unbiased problem instance

3.2.1   STRIPS/PDDL

- Block World

- Classical Planning Domains (IPC problems collection)

3.2.2 HATP

- Assisted assembly

- Dock-worker robot

3.2.3 ASP

- Evolutionary studies

- Natural language processing: Dependency parsing, linguistics, taxonomy, inferring Phylogenetic Trees

- Graphs: graph colouring, Large clique, Hamiltonian cycle

- Decision Support for the Space Shuttle

- Automated Product Configuration

- Coaches

3.2.4 MDP

- SPUDD: grid world, coaches

- Often in combination with other formalisms, any task with probabilistic outcomes (perception/state)

3.2.5 POMDP

- grid world with probabilistic outcomes (faulty navigation)

3.2.6 Various

Waitress Robot

A mobile Robot with wheels and two arms has to serve clients in a restaurant environment. One of the arms is a tray where he can put 3 food/cocktail at maximum. He has also sensors to perceive the environment. He transports

the goods from the kitchen to the clients, taking ordinations from them and welcoming incoming persons. After he has received a client, he must bring him to the table and take the ordination. He could take multiple ordinations, but he must reply sooner to the cook when a food is ready. Every food has its maximum delivery time before it gets cold (discrete values from 1 to 3 minutes). Cocktails have no delivery time. People could also recall the attention of the robot by the means of a button on the table so that they can order more foods/drinks. So the task priority is:

1. deliver goods

2. welcome person and take ordination

3. take random ordinations

An object on the tray could fall down with a probability 0.4, so every time the robot deliver goods he must check if the object is effectively on the tray. If not he must notify the kitchen. The kitchen and the main room are connected via a door that could be both closed or open with equal probability and the robot must check its condition each item. Due to the ambient condition, he could also fail any perceive action with probability 0.2. If the door is closed and the robot doesn't perceive it, all the goods will fall certainly (if is there are any). The goal is to minimize the waiting time both for the clients and the cook given the previous priority list.

Assisted Assembly

This example is inspired by the one presented in [2].
In assisted assembly task a robot must bring tools to a worker and helps him with a welding procedure. The robot is a humanoid, thus with two arms and two legs, and has a perception system. The environment contains cavities that only the robot could pass in order to reach the worker or move a particular tool from one room to another. The worker is standing in front of its working table. The worker access is working station through a manhole that remains open all the time, costituting a threat for the robot. If the robot cross all the the worker room he could fall down in the hole with probability

0.6. Tools are placed randomly in other rooms and the robot knows exactly where they are through a system of external cameras. When the worker requires a tool, he must bring him as soon as a possible minimizing the traveled rooms. There are 8 rooms and they are all accessible through the cavities. Cavities requires that the robot assumes particular configurations in order to pass through. Each configuration of the robot could be evaluated by a predicate that tells in constant time if the robot will pass or not, but this valid configurations changes according to the number and the type of tools transported. The robot has N degrees of freedom. For the assembly part of the task, the input of the problem is the kinematic model of the robot with its initial configuration, the representation of the environment and the tools required by the worker. The output is all the configurations that the robot must assumes in order to full fill the requirements, that are brings all the tools and minimize worker waiting time.

In the assisted welding the robot stays in the same room of the worker and, since he has better understanding of spatial geometry, he must indicate to the worker the exact trajectory of the welding. In this case the input of the problem are the initial configurations of both the robot and the worker, and the output must be a sequence of configurations that minimizes the change in stance of the worker according to some comfort measurement parameter that evaluates worker configurations.

COACHES

This examples are taken from [1].

Basic Service Robot
A robot is able to execute two activities A and B upon request of the user, but the user does not know about these robot abilities. The robot is normally in a waiting state. Whenever a user is detected in front of the robot (for example, with a simple face detector procedure), the robot can decide whether to start an interaction (for example, through a spoken dialogue)

and, if so, it describes the activities that it can do A and B. The user can now select one of them or answer that s/he is not interested (for example by selecting an option on a user interface or by a speech command). If a goal is selected, the robot executes a task for achieving it. Each goal can be achieved with two alternatives tasks *TaskA1* and *TaskA2* for *A* and *TaskB1* and *TaskB2* for *B*. After the execution of the task, the robot comes back to the wait state, waiting for a next user. During the execution, the following inconveniences may occur: the user does not complete the interaction (for example, s/he does not answer the robot question), any task may fail for reasons not modeled in the task description, and any task may be aborted according to some external command or condition.

Service Robot assisting customers of a shop-ping mall
A robot operates in the corridor of a shopping mall including several shops. Each shop-keeper can define a set of advertisements that the robot is asked to communicate to customers of the mall and each advertisement has an associated reward for the shopping mall. The robot can execute the following actions in the environment: move to any location (i.e., in front of any shop), approach a specific person or a group of people, perform advertisement actions, perform other assistive actions upon request, bring a person to any location in the mall. The problem we want to solve is the following: given the current position of the robot, the information about shops and advertisements and an initial probability distribution of the presence of people in the shopping mall (for example, provided by an external system of video-cameras), plan and execute the behavior that maximizes the global actual reward, also recovering from possible failures due to unpredicted or erroneous interactions with people.

Chapter 4

Foundation Tools

4.1 Petri Net Plan

We need a language that can represent non-instantaneous and conditional actions.

Petri Net Plans (PNP)[21][22] is a formalism for represent complex plans using an high-level description. It is inspired by languages for reasoning about actions, like the ones presented in 3.1, but it more expressive and offer all the operators needed for representing conditional plans, concurrent actions and even more (multi-agent plans, for instance).

They also offer non-deterministic choice operator for representing non-deterministic choice action, useful in learning algorithms.

The execution of a PNP is very efficient, and allow the design of real-time and active behaviours. Some examples of PNPs applications are [23][24][25].

4.1.1 Formal Definition

A PetriNet Plan is a PetriNet augmented with a set of goal markings G. More formally, it is a tuple $< P, T, F, W, M_0, G >$ where:

- P $= \{p_0, ..., p_n\}$ is a finite set of places.

- T $= \{t_0, ..., t_n\}$ is a finite set of transitions.

- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.

- $W : F \Rightarrow 1, 2, 3, ...$ is a weighting function.

- $M_0 : P \Rightarrow 0, 1, 2, ...$ is the initial marking.

- $P \cup T \neq 0$ and $P \cap T = 0$

Petri Nets represent the structure of a system as a directed, weighted and bipartite graph with two type of nodes: places and transitions. With this tools we can model complex systems in terms of their internal state and its changes.
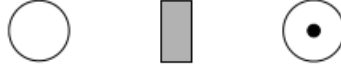

Figure 4.1: (1) a place. (2) a transition. (3) a place with one token.

Places represent the execution phases of an action, while transitions represent events. In particular each action is represented by three states: initial, execution and final state. Between them there are action starting transition, action terminating transition and there might be some action interrupts and/or control transitions.

The evolution of the Petri Net is described by the firing rule: a transition is enabled when the number of tokens for its initial places is equal at least to the number of tokens of the edge weight. When the transition fires, a number of tokens equal to the weight of the input edges is removed from the initial places and a number of tokens equal to the output edges is added to output nodes. Petri Net Plans are defined in terms of actions and operators. There are three types of actions:

1. **no-action**. is a PNP with one place and no transition.

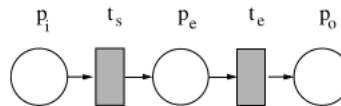2. **ordinary-action**. is a PNP with three places and two transitions, as described before.

Figure 4.2: an ordinary action.

3. **sensing-action**. this are used when the action involve a property to evaluate at run time, and places/transitions are defined accordingly.
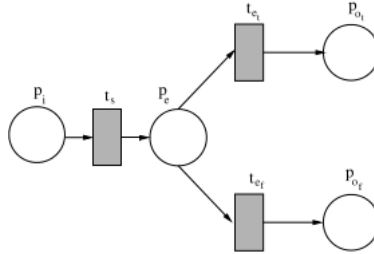

Figure 4.3: a sensing action.

Actions could be combined in order to achieve complex behaviours. These are called operators.

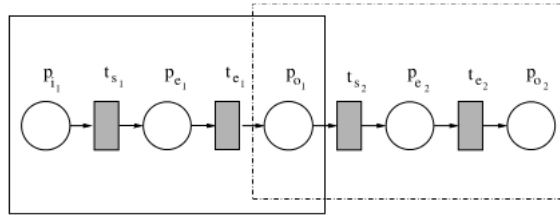- **sequence**. allows the execution of two actions in sequence.

Figure 4.4: sequence operator.

- **conditional**. combining sensing action with ordinary actions is possible to obtain conditional structure.
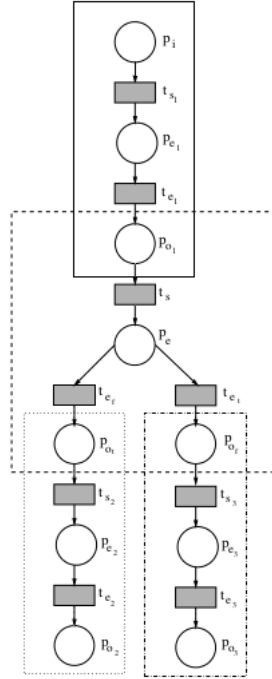
Figure 4.5: conditional operator.

- **interrupt**. allows to interrupt the execution of an action, if some condition is met, and to continue the execution on a new branch.

- **loop**. using an interrupt, is possible to define action that iterates until a condition is true.
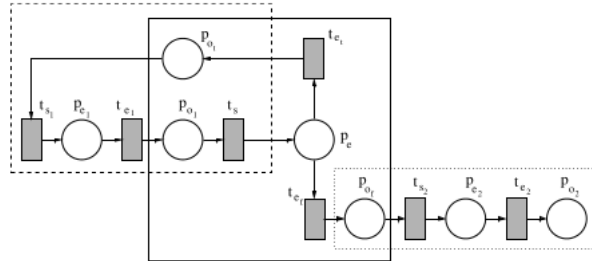

Figure 4.6: loop operator.

- **fork/join**. allows concurrency. fork generates multiple threads starting from one, while join allow the synchronization of multiple threads
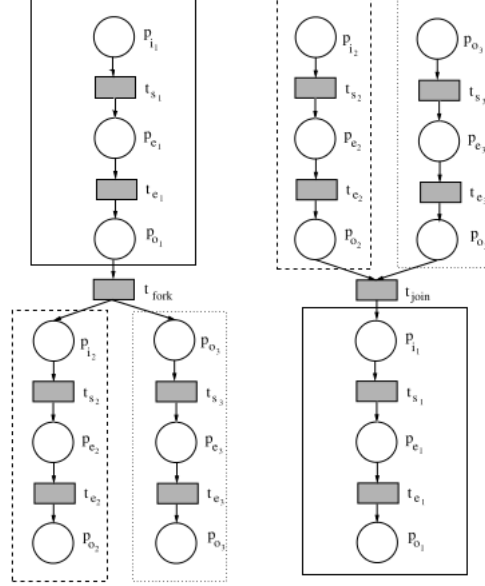
into one.

Figure 4.7: (left) fork operator. (right) join operator.

In the following page there is an example of a complete PNP, with an interrupt defined during the action *followperson*:

4.2 Robot Operating System (ROS)

"Robot Operating System (ROS) is a collection of software frameworks for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages." [1]

ROS was originally developed at Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR (STanford AI Robot) project. From 2008 to 2013 the development take place at Willow Garage, a robotics research institute. Finally, from 2013, ROS become part of the Open Source Robotics Foundation.
During all this time, ROS has growth in importance and diffusion.
In fact, before ROS advent, each laboratory or research group had to write down from scratch specific code for his platform and for every task, even the more mundane. This not only is a time expensive task but restrains the diffusion of new algorithms and ideas. With the layer of abstraction offered by ROS, it is possible to write nodes for common tasks and specific hardware that are reusable and so more easy to diffuse. For instance, ROS offers packages for perception, object identification, segmentation and recognition, face recognition, planning, grasping, localization, mapping and much more. The main limitation of ROS resides in its latency since is built on top of Linux and is not a real-time OS. This led to the born of ROS 2.0, which support real-time operations, and RT-ROS, which is natively real-time.

---

[1]https://en.wikipedia.org/wiki/Robot_Operating_System

4.3 ROSPlan

Chapter 5

Conditional Planning and Execution

5.1   Our Approach

- world/agents state

- environment representation/map (semantic/topology)

- action (preconditions/effects)

- interaction (if social)

- time (explicit/implicit)

Different planning formalisms try to capture different aspects of real-world problems. By the way they model and abstract the world, they also produce different solution concepts.

Is not always obvious in advance which formalism is the best for a specific problem. Less expressive formalism are often preferred due to the fact that are less expensive in terms of computational power.

For instance if we consider decidable logic formalisms, $ALCK_{NF}$ is one of the more expressive and yet the planning in general it's included in PSPACE, so too complex to run on a real robot on realtime.

Classical Planning models only deterministic actions, and the planner is often

embedded in a controlled execution loop that replan on unexpected outcomes. In fully-observable and non-deterministic planners, actions are assumed to be non-deterministic but there is no uncertainty about the perception. They produce strategies that are guaranteed to reach the goal.

If we expand the domain and consider also probabilistic planners, they introduce also outcome probabilities and the aim, in this case, is to maximise a cumulative reward. They are also the most expensive and often intractable at representation and/or execution level. This brief analysis shows that there is a compromise between expressivity and computational complexity.

Rule-bases are not manageable when they become larger. This rules are created by experts that express their knowledge through the rules and it is difficult to find all the inconsistencies and the side-effects when new rules are inserted. Epistemic logic could helps us to deal with all the uncertainties the real world introduce in the domain of a problem but, again, it is very likely that the state space grows exponentially. A solution could be to decompose the state in subgroups and use separate approach for each group. If we assume that the only strong uncertainty comes from the state of mind of the human agent, we could think to subdivide the state and combine different formalisms in order to deal with each group.

A possible subdivision could be between environmental and human state variables:

$$|Y| + |2^Z| << |2^X| \text{ and } X = Y \cup Z$$

The next step is to identify the desirable characteristics of each formalism for each of the task benchmarks we have defined in 3.2.

This is done in the following table:

- Huge state space

- Decomposable

- Hard Synchronization

- Non-deterministic observation

- Non-deterministic outcomes

Summarizing, our aim is to define an architecture capable of combine different formalisms in order to satisfy tasks requirements, while finding the best trade-off between complexity and expressivity.

In the next section I will use the examples in 3.2 to describe possible domains using the formalisms presented in 3.1.
In particular, I will characterize the input/output of each problem and provide a sketch for a possible optimal solution.

5.2 Motivating Example

5.2.1 Example: Service Robot assisting customers of a shopping mall

This example is taken from [1].
The input to the problem is:

- a qualitative information about the location of the robot and the person

- an information indicating if the person is moving or if he stop for a certain time

- the advertisements of the shop

The output is the executed behaviour which maximizes the global actual reward, which can be defined depending on the present needs (advertise the major number of people, save energy by not wandering too much around, etc). The robot could execute some actions and provide a some services. These are:

- move to a location

- approach a person or a group of people

- perform advertisement

- bring a person to a location

- help people carry things

For the sake of clarity, we introduce a simple example in which we focus on the aspects of the interaction and we do not consider other uncertainties coming from the robotic system (issues in localization and navigation, face detection, etc). Consider the situation in the following image:
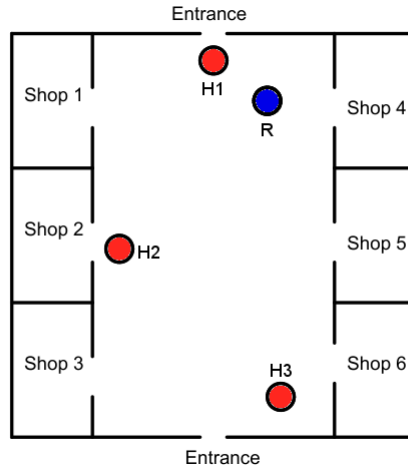

Figure 5.1: A snapshot of the mall, where R is the Robot and H_i a person

The qualitative information about the people locations indicates that there is a person in front of shop 2, another wandering nearby shop 6 and a third person in front of him. So the set of actions could be represented as

$$A = \{sense\_person, move, bringto, approach\_person, approach\_group, advertise, carry\_bags, wait, ask\_clothes, ask\_food, ask\_object, bye\}$$

where X could be a generic location or the position of a shop. The robot is able to represent knowledge about the environment and try to guess people wishes. For instance, based on the people locations and information like the time, he tries to infer from its knowledge if a person is interested in one particular shop or services and could provide an advertisement. He maintains an internal database of all the goods for sale, divided by category, so he can better individuate people needs. If we recap the formula introduced in 5.1,

37

we have that:

$$X = Y \cup Z$$

where X represents the state variables, Y are the environmental state variables while Z are the human state variables.

$Z = \{food, clothes, object, WC, help, none\}$
$Y = \{time, person\_atX, group\_atX, RLocation, foodCategory_i, genericGood_j,$
$clothingStore_k\}$

where $food, clothes, WC, help, none$ represent the person interest in a service, while $FoodCategory_i, GenericGood_j, ClothingStore_k$ represents the specific required goods.

Consider person $H2$, that is standing in front of shop 2.

Lets assume that

- shop 2 and shop 4 are clothes store. The first is a sportive clothes store, while the second is an high fashion store.

- shop 3 is a WC

- shop 1 is a market

- shop 5 and shop 6 are a chinese and an italian restaurant.

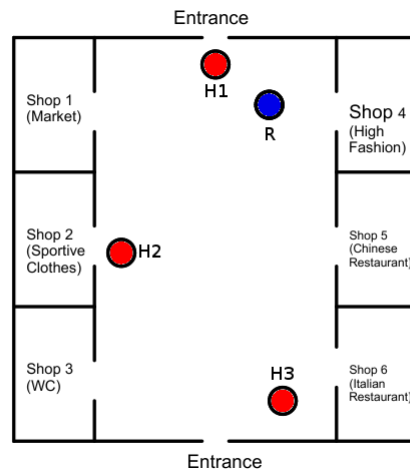This situation is summarized in the following figure:

Figure 5.2

We can reasonably think that the person is looking for a specific article inside this shop, or another clothing shop, and that he's not certainly hungry.
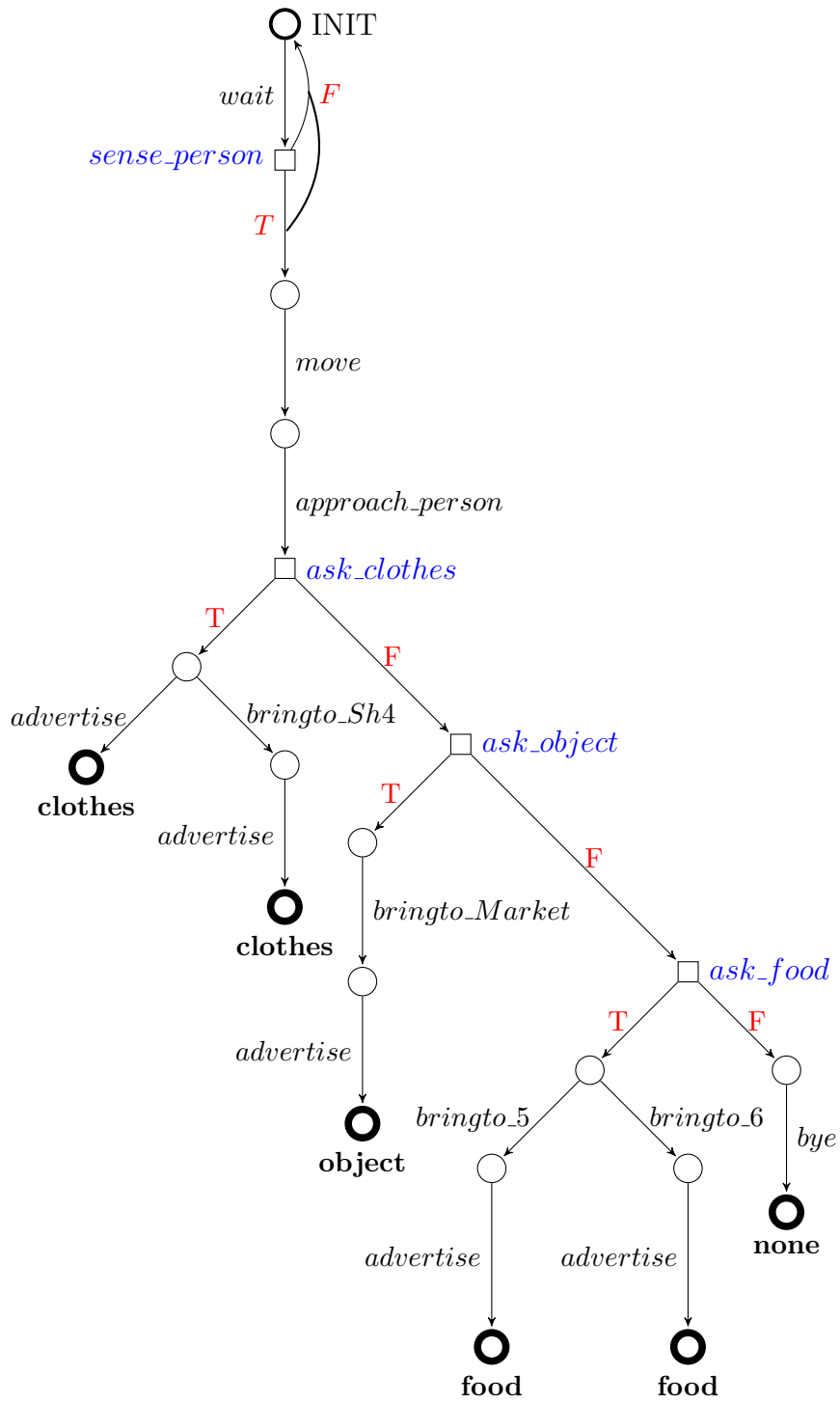So a possible plan for $H2$ would be:

Figure 5.3: a plan for H2

Where the squares (□) represents "sensing actions", that are non determin-
istic actions used by the robot in order to investigate about environment
properties. We assume that a state in which at least one of the Z properties
holds is a goal state.

I will now show possible solutions for this problem using some of the intro-
duced formalisms.

PDDL

First, I will present a solution in PDDL (ver. 1.2), which is a standard "de
facto" for the specification of planning problems.

In order to do that, we must characterize objects, predicates, actions, initial
and final states.

I have used FF (ver. 2.3) [20] planner in order to produce the planes and
check the satisfiability of the problem. In this domain description I will con-
sider only H_2 for the sake of clarity

For the predicates we have:

```
(:predicates
          ;robot and person properties
          (person ?x) (robot ?x)
          (at-robot ?x) (at-person ?x)
          (close-to ?x)(wait ?x)(bags-loaded ?x)

          ;needs represents the person necessity
          (needs-help ?x) (needs-WC ?x) (needs-obj ?x)
          (needs-food ?x) (needs-clothes ?x)

          ;desire predicates used to disambiguate which food/cloth
          (desire-italian ?x) (desire-chinese ?x)
          (desire-sportive ?x) (desire-fashion ?x)

          ;locations predicates
```

```
          (SHOP ?x) (entrance ?x) (WC ?x) (market ?x)
          (sportive-clothes ?x) (fashion-clothes ?x)
          (chinese-restaurant ?x) (italian-restaurant ?x)

          ; 'sensing' actions
          (ask-italian ?p) (ask-chinese ?p)
          (ask-sportive ?p) (ask-fashion ?p)
          (ask-obj ?p) (satisfied ?x)

)
```

I will not enlist all the actions specifications, but instead I will focus on sensing actions:

```
(:action sense-person
                  :parameters ( ?r ?p)
                  :precondition (and(person ?p)(robot ?r)(wait ?r))
                  :effect (not(wait ?r))
)

(:action ask_clothes
        :parameters ( ?r ?p ?x)
        :precondition (and (person ?p)(robot ?r)(close-to ?p))
        :effect (and
                    (needs-clothes ?p)
                    (when
                         (and (at-person ?x)(sportive-clothes ?x))
                         (desire-sportive ?p)
                    )
                    (when
                         (and (at-person ?x)(fashion-clothes ?x))
                         (desire-fashion ?p)
                    )
                )
)
```

```
(:action ask_food
            :parameters ( ?r ?p ?x)
            :precondition (and (person ?p)(robot ?r)(close-to ?p)(at-person ?x)
            :effect (and
                        (needs-food ?p)
                        (when
                            (italian-restaurant ?x)
                            (desire-italian ?p)
                        )
                        (when
                            (chinese-restaurant ?x)
                            (desire-chinese ?p)
                        )
                    )
)

(:action ask_object
            :parameters ( ?r ?p ?x)
            :precondition (and (person ?p)(robot ?r)(close-to ?p)(at-person ?x)
            :effect (needs-obj ?p)
)
```

It is clear from the specifications that they are not real sensing actions. In fact
they plan a priori the human agents needs and there are no not-deterministic
effects.

The objects are:

```
(:objects shop1 shop2 shop3
        shop4 shop5 shop6
        entr1 entr2
        food clothes obj
        WC help none
        r h2
  )
```

where $\{food, clothes, obj, WC, help, none\}$ models the Z variables presented in 5.2.

We can then characterize the initial and final state as:

```
(:init (robot r)(person h2)
       (wait r)
   (at-robot shop4) (at-person shop2)
       (entrance entr1)(entrance entr2)
       (SHOP shop1)(SHOP shop2)(SHOP shop3)
       (SHOP shop4)(SHOP shop5)(SHOP shop6)
       (market shop1)(WC shop3)
       (sportive-clothes shop2)(fashion-clothes shop4)
       (chinese-restaurant shop5)(italian-restaurant shop6)
)

(:goal
    (satisfied h2)
)
```

The goal is thus to satisfy the human needs.

PDDL was meant to express the "physics of a domain, and not instrisic properties of an agent. In particular, in order to implement sensing actions, we must extend PDDL with the concept of knowledge. An example can be found in [17], in which the authors have introduced in the PDDL action definition the notion of inertial and sensing effects.

KPDDL

Chapter 6

System Overview

PNPgen

PNPgen[26] is a library included the Petri Net Plans distribution responsible for the automatic generation of PNPs from a formal definition provided.
This is usually used in conjunction with a planner, which output is interpreted and executed as a PNP.
The main advantages of this approach are two:

1. The produced plan can be executed and monitored through the PNP execution engine and easily integrated into ROS thanks to the PNP-ROS bridge.

2. Once produced,the PNP can be extended of modified to deal with unexpected outcomes that may rise during the execution of an action and not considered in the domain, either manually or automatically as we will discuss later.

   The languages considered are two:

   - PDDL (FF planner): this produce a linear plan.

   - KPDDL (kplanner): this produce a conditional plan, or more generally a DAG (direct acyclic graph).

   - ROSPlan (modified kplanner): similar to KPDDL, produce a tree.

Bibliography

[1] L. Iocchi, L. Jeanpierre, M. T. Lazaro, A.-I. Mouaddib
A Practical Framework for Robust Decision-Theoretic Planning and Execution for Service Robots 2006.

[2] Rachid Alami, Mamoun Gharbi, Benjamin Vadant, Raphäel Lallement and Adolfo Suarez
On human-aware task and motion planning abilities for a teammate robot

[3] Richard E. Fikes, Nils J. NHsson
STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving

[4] Tom Bylander
The Computational Complexity of Propositional STRIPS Planning. Artificial Intelligence 69 (September 1994)

[5] Pednault
Formulating multi-agent dynamic-world problems in the classical planning framework.

[6] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins
PDDL - The Planning Domain Definition Language

[7] J. McCarty, P. Hayes
Some philosophical problems from the standpointof artificial intelligence, Machine Intelligence 4, 1969

[8] R. Reiter
Proving properties of states in the situation calculus, Artificial intelligence 64, 1993

[9] Vladimir Lifschitz
What Is Answer Set Programming?

[10] ASP
`https://en.wikipedia.org/wiki/Answer_set_programming`

[11] Esra Erdem, Erdi Aker, Volkan Patoglu
Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution

[12] Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie
Developing High-level Cognitive Functions for Service Robots

[13] Yi Wang, Joohyung Lee
Handling Uncertainty in Answer Set Programming

[14] Francesco M. Donini, Daniele Nardi, Riccardo Rosati
Autoepistemic Description Logics

[15] Hakan L. S. Younes, Michael L. Littman
PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects

[16] Scott Sanner
Relational Dynamic Influence Diagram Language (RDDL): Language Description

[17] Luca Iocchi, Daniele Nardi, Riccardo Rosati
A PDDL Extension for Describing Planning Domains with Incomplete Information and Sensing

[18] Nicholas Roy, Geoffrey Gordon, Sebastian Thrun
Finding Approximate POMDP Solutions Through Belief Compression

[19] Ronen Brafman, Alexander Gorohovski, Guy Shani
A Contingent Planning-Based POMDP Replanner

[20] J. Hoffmann *FAST-FORWARD*
`https://fai.cs.uni-saarland.de/hoffmann/ff.html`

[21] Vittorio Amos Ziparo and Luca Iocchi.
Petri Net Plans.
In Proc. of Fourth International Workshop on Modelling of Objects, Components, and Agents (MOCA'06) @ Petri Nets 2006 and ACSD 2006, Turku, Finland, 2006.

[22] V. Ziparo, L. Iocchi, D. Nardi, P. Palamara, and H. Costelha.
PNP: A formal model for representation and execution of multi-robot plans
In Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), 2008.

[23] E. Bastianelli, D. Bloisi, R. Capobianco, F. Cossu, G.Gemignani, L. Iocchi, D. Nardi.
On-line Semantic Mapping.
In 16th International Conference on Advanced Robotics, 2013.

[24] P. Palamara, V. Ziparo, L. Iocchi, D. Nardi, P. Lima, H. Costelha.
A robotic soccer task using Petri Net Plans.
In Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS), 2008.

[25] A. Farinelli, L. Iocchi, D. Nardi, V. A. Ziparo
Assignment of Dynamically Perceived Tasks by Token Passing in Multi-Robot Systems.
Proceedings of the IEEE, Special Issue on Multi-Robot Systems, 2006

[26] F. M. Carlucci, L. Nardi, L. Iocchi, D. Nardi.
Explicit Representation of Social Norms for Social Robots.
In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2015.