CSE 513T Theory of Artificial Intelligence & Machine Learning Nov 17, 2016

Lecture 23

Lecturer: Brendan Juba *Scribe: Ying Wang*

1 Today's Topics

1. Finish Logical Encoding of Actions and Change, Planning-As-Satisfiability
2. Markov Decision Processes and PAC rule.
3. Policies and Fault Tolerance

2 From last time...

Encoding of STRIPS environments

1. Actions with preconditions
   action α at time $t \longleftrightarrow$ literal $a_{\alpha,t}$,

   $$[a_{\alpha,t}] \Rightarrow e_{1,t+1} \; [a_{\alpha,t}] \Rightarrow P_t$$

   $$...$$

   $$[a_{\alpha,t}] \Rightarrow e_{r,t+1} \; [a_{\alpha,t}] \Rightarrow P_t$$

   e is the effect of α, P is the precondition of α.

   For each conditional effect e' of α at time t,

   $$[q_{1,t} \wedge ... \wedge q_{s,t} \wedge a_{\alpha,t}] \Rightarrow e'_{t+1}$$

2. "Unique" action at each step

   $$\bigvee_{\alpha \in A} a_{\alpha,t}, \; t = 1, 2, ..., T (\geq 1 \text{ action})$$

   $$\bigwedge_{\alpha \neq \alpha' \in A} \neg a_{\alpha,t} \vee \neg a_{\alpha',t}, \; t = 1, 2, ..., T (< 2 \text{ actions})$$

3. "Frame axioms"
   In STRIPS, only changes in environment are due to actions. One simple natural encoding: using naf, for every $l, t = 1, ..., T$,

   $$[l_t \wedge \text{not}(\neg l_{t+1})] = l_{t+1}$$

   without naf,

   $$(\forall l)[\neg l_t \wedge l_{t+1}] \vee_{\alpha : l \, effect} a_{\alpha,t}$$

   more compact, one clause per literal

   $$\neg[\neg l_t \wedge l_{t+1}] \vee_{\alpha : l \, effect} a_{\alpha,t}$$

Proposition 1 *For any time horizon T,*

$$\bigwedge_{t=1}^{T}([\textit{unique } \alpha_t] \wedge [\textit{preconditions}]_{\alpha \in A} \wedge [\textit{effects}]_{\alpha \in A} \wedge [\textit{frame axions}]_{l,\alpha \in A})$$

(where $l_{1,1} \wedge ... \wedge l_{n,1}$ has as its unique satisfying assignment for each T-step plan $\alpha_1, ..., \alpha_T$, and in this satisfying assignment for each $t = 1, ..., T$, $x_{1,t}, ..., x_{n,t}$ contain the state of the environment after the actions $\alpha_1, ...\alpha_t$.)

Corollary 2 *By adding a conjunction $G(x_{1,T}, ...x_{n,T})$, we obtain a formula that is satisfiable iff there is a T-step plan that achieves G.*

Corollary 3 *The settings of $a_{\alpha,t}$ encode the plan that achieves G in T steps (at step t, take the action s.t. $a_{\alpha,t} = 1$ in this satisfying assignment).*

Proof (Sketch)By induction on t (base trivially satisfied at $t = 1$), given true for the $t-1$ step plan $\alpha_1, ..., \alpha_{t-1}$, the plan with action α at time t corresponds to setting $a_{\alpha,t} = 1$ and $a_{\alpha',t} = 0 \; \forall \; \alpha' \neq \alpha$. This attribute can be set iff the preconditions of α were satisfied in the unique satisfying assignment at time $t-1$. Then the frame and effect axioms uniquely determine the settings of the environment variables. ■

Note in general, STRIPS planning is NP-hard, so we can't hope for much better. (exercise: come up with a domain, e.g, where the agent's task is to find a satisfying assignment to a CNF)

So that's classical planning. But classical planning makes two pretty strong assumptions.

Strong assumptions in STRIPS plannning

1. Environment is deterministic and responds predictably to actions.
2. Complete information

If we relax the first assumption, we get (factored(meaning states are decompose into attributes)) Markov Decision Process(MDP), so we make the transitions probabilistic. The general MDP says I don't know attributes, I only know states, which is harder. So we only focus the factored case. If we relax the second assumption, we end up with Partially Observed Markov Decision Process(POMDP). The MDP requires observing the environment and responding to it. The POMDP requires reasoning about the possible states of the environment. Both the two assumptions are unrealistic, being able to learn a planning model is going to require everything we did this semester.

3 Markov Decision Process(MDP)

We assume that the distribution over states at time $t+1$ depends only on the actual outcome of the current state and the action we choose at time t.

$$Pr[\overrightarrow{x_{t+1}}|\alpha_t, \overrightarrow{x_t}, \alpha_{t-1}, \overrightarrow{x_{t-1}}, ..., \alpha_1, \overrightarrow{x_1}] = Pr[\overrightarrow{x_{t+1}}|\alpha_t, \overrightarrow{x_t}]$$

(independent of the history) "factored MDP": the states can be desribed by n attributes.

In general, MDPs can encode highly unpredictable environments, e.g., shuffling a deck of cards, $x_t \leftrightarrow$ order of cards in the deck, $x_t \rightarrow x_{t+1}$, sample an independent, uniformly random ordering. More structured MDPs will be our focus. For example, Robot and box STRIPS domain, rules describing actions, e.g. $[\text{pick_up}]_{i,t} \Rightarrow \text{holds}_{r,b,t+1}$.

- might only be $1-\epsilon$-valid for some $\epsilon > 0$ for $t = 1, 2, ...$, i.e, "pick_up" might fail to work w.p $\epsilon > 0$.
- Frame axioms might also fail w.p $\epsilon > 0$, if some other robot moves the box etc.

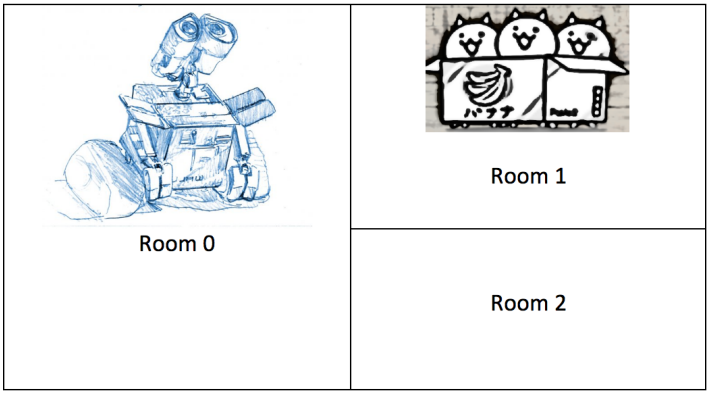
4 Policies & Fault-tolerance

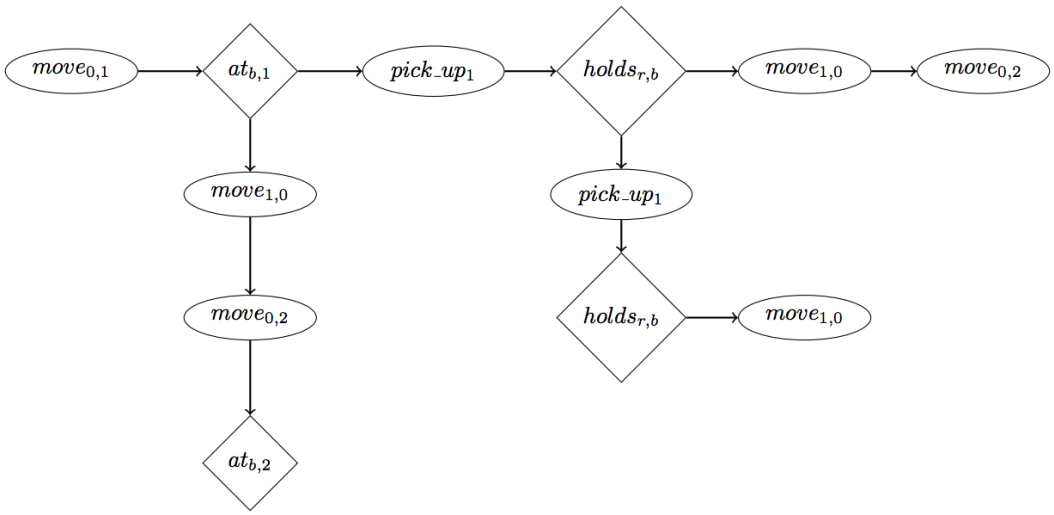
It no longer suffices to blindly follow "a sequence of actions". We require a policy:

$$\Pi : \mathcal{X} \times (\mathcal{A} \times \mathcal{X})^* \to \mathcal{A}$$

where Π is the policy, $\mathcal{X}$ is the current states, $\mathcal{A}$ is the choice of next action. Natural kind of policy decision tree, we'll take the convention that our T-horizon policy is given by a single tree (generating a plan).

Example: clumsy robot policy





$move_{1,0}$ and $pick_up_1$ are "Recovery actions" in response to rules failing ("faults"), which increases probability that plan succeeds.

e.g. suppose that the probability that each rule fails is $p < 1$, and failures are independent. Probability of more than k faults decays exponentially with probability$\leq p^{k+1}$. So if I want to guarantee success w.p $1 - \epsilon$, then I only need to handle $k \sim \frac{\log \frac{1}{\epsilon}}{\log \frac{1}{p}}$ faults.

A "policy" that handles 0 faults: a classcical plan, is just a single branch of the tree, known as a "conformant" policy.

A "policy" that handles k faults: is one that checks after each action to see if the environment is in the expected state, and executes a policy that handles $k-1$ faults if not. Otherwise, it continues with a policy that handles k faults.

So at each of these branching nodes, at most one branch needs to continue to handle k faults.

So k-fault tolerant decision tree policies are given by rank-k tree. (if we treat the actions as just labeling branches). We'll be able to find such policies efficiently.

5 Next time:

POMDPs. In particular, factored states with "observation" given by a masking process.(i.e the states attributes are hidden in the observation) e.g. $a_{t,b,i}$ only observed if $a_{t,b,i} = 1$.