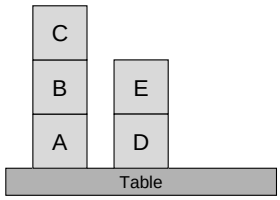
Module Title: PLAN
Exam Diet (Dec/April/Aug): April 2007
Brief notes on answers:

1. (a) **AI Planning**: We (humans) usually only plan our actions when this is necessary. Why is this? Under what circumstances do humans act with and act without explicit prior planning?

   Answer:
   Planning is complicated and time-consuming (trade-off: cost vs. benefit). Possible circumstances:

   - acting without (explicit) planning:
     - when purpose is immediate
     - when performing well-trained behaviours
     - when course of action can be freely adapted
   - acting after planning:
     - when addressing a new situation
     - when tasks are complex
     - when the environment imposes high risk/cost
     - when collaborating with others

   (b) **Situation Calculus**: Representations of planning domains and problems usually include descriptions of world states and activities. The figure below shows a world state in the classic Blocks World. Give a logical formula Σ_{s_i} that would describe this state as an initial state in the situation calculus.



   Answer:

$$\begin{aligned}\Sigma_{s_i} =& on(A, Table, s_i) \wedge on(B, A, s_i) \wedge on(C, B, s_i) \wedge \\ & on(D, Table, s_i) \wedge on(E, D, s_i) \wedge \\ & clear(C, s_i) \wedge clear(E, s_i) \wedge clear(Table, s_i)\end{aligned}$$

   (c) **State-Space Search**: In classical planning, a planning problem is solved by searching for a solution plan. Define, in pseudo-code, the non-deterministic ground backward state-space search algorithm for a given statement of a STRIPS planning problem $P = (O, s_i, g)$.

   Answer:

function $groundBwdSearch(O, s_i, g)$
$subgoal \leftarrow g$
$plan \leftarrow \langle\rangle$
loop
if $s_i.satisfies(subgoal)$ then return $plan$
$applicables \leftarrow$ {ground instances from O relevant for $subgoal$}
if $applicables.isEmpty()$ then return failure
$action \leftarrow applicables.chooseOne()$
$subgoal \leftarrow \gamma^{-1}(subgoal, action)$
$plan \leftarrow \langle action \rangle \bullet plan$

(d) **Plan-Space Search**: In plan-space search the nodes in the search space are partial-order plans which contain explicit causal links between the different actions in the plan. Thus, planners that perform a plan-space search must find the new threats in a partial plan when the plan is refined. For which types of refinement do the threats need to be detected? For each of these describe in pseudo-code how the detection is performed.

Answer:

- in the initial plan π_0: no threats
- when adding an action a_{new} to $\pi = (A, \prec, B, L)$:
  for every causal link $\langle a_i \xrightarrow{p} a_j \rangle \in L$
  if $(a_{new} \prec a_i)$ or $(a_j \prec a_{new})$ then next link
  else for every effect q of a_{new}
  if $(\exists \sigma : \sigma(p) = \sigma(\neg q))$ then q of a_{new} threatens $\langle a_i \xrightarrow{p} a_j \rangle$
- when adding a causal link $\langle a_i \xrightarrow{p} a_j \rangle$ to $\pi = (A, \prec, B, L)$:
  for every action $a_{old} \in A$
  if $(a_{old} \prec a_i)$ or $(a_j = a_{old})$ or $(a_j \prec a_{old})$ then next action
  else for every effect q of a_{old}
  if $(\exists \sigma : \sigma(p) = \sigma(\neg q))$ then q of a_{old} threatens $\langle a_i \xrightarrow{p} a_j \rangle$

(e) **STN Planning**: Yet another search space is searched by STN planners. Here, a node in the search space is a task network. A possible plan refinement operator selects a task t in the network and chooses an applicable and relevant method m to decompose t. Give definitions for:

- applicability of a method instance m,
- relevance of a method instance m, and
- the decomposition function δ.

Answer:

- A method instance m is applicable in a state s if
  - $precond^+(m) \subseteq s$ and $precond^-(m) \cap s = \emptyset$.
- A method instance m is relevant for a task t if
  - there is a substitution σ such that $\sigma(t) = task(m)$.
- The decomposition of a task t by a relevant method m under σ is
  - $\delta(t, m, \sigma) = \sigma(network(m))$ or
  - $\delta(t, m, \sigma) = \sigma(\langle subtasks(m) \rangle)$ if m is totally ordered.

(f) **SAT-Based Planning**: The idea behind SAT-based planning is quite similar to the idea used in the situation calculus: the planning problem is reformulated as a theorem proving problem. Show how the initial state s_i and the goal g of a propositional planning problem $P = (A, s_i, g)$ can be represented as part of the SAT encoding of a bounded planning problem, i.e. what are the propositional formulas that represent the initial state and the goal?

Answer:
Let F (the fluents) be the set of all the proposition symbols that occur in $P = (A, s_i, g)$ (in preconditions, positive and negative effects, in the initial state, or in the goal) and let n be the length of the plan sought (bounded planning problem). Then the initial state can be represented by the formula:

$$\bigwedge_{f \in s_i \subseteq F} f_0 \wedge \bigwedge_{f \in (F - s_i)} \neg f_0$$

and the goal can be represented by the formula:

$$\bigwedge_{f \in g^+} f_n \wedge \bigwedge_{f \in g^-} \neg f_n$$

(g) **Temporal Planning**: During the planning process, explicit time constraints can be managed by a constraint manager, or we can use temporal operators. In the latter case, a temporal database contains a finite set of temporally qualified expressions (*tqe*s). Consider the following set $\mathcal{F}$ of *tqe*s:

- $at(r_1, loc_1)@[t_0, t_1[$,
- $at(r_2, loc_2)@[t_0, t_2[$,
- $at(r_2, path)@[t_2, t_3[$,
- $at(r_2, loc_3)@[t_3, t_4[$,
- $free(loc_3)@[t_0, t_5[$,
- $free(loc_2)@[t_6, t_7[$

Show that the following *tqe* can be supported by $\mathcal{F}$ by listing the possible enabling conditions?

- $at(r_2, l)@[t_b, t_e[$

Answer:

- $t_0 \leq t_b \wedge t_e \leq t_1 \wedge r_2 = r_1 \wedge l = loc_1$ or
- $t_0 \leq t_b \wedge t_e \leq t_2 \wedge r_2 = r_2 \wedge l = loc_2$ or
- $t_2 \leq t_b \wedge t_e \leq t_3 \wedge r_2 = r_2 \wedge l = path$ or
- $t_3 \leq t_b \wedge t_e \leq t_4 \wedge r_2 = r_2 \wedge l = loc_3$

2. (a) **AI Planning**: Planning is an area that has been researched in Artificial Intelligence for a long time now. What do we mean by "AI Planning"? What is being studied in this field?

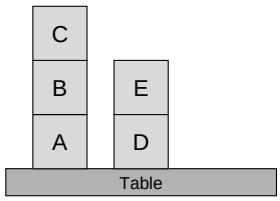
Answer:

- "planning":
  - explicit deliberation process that chooses and organizes actions by anticipating their outcomes (reasoning about actions)
  - aims at achieving some pre-stated objectives
- "AI planning":
  - computational study of this deliberation process

(b) **Situation Calculus**: A problem that was encountered by planners early on is the so-called frame problem. Consider a theory in the situation calculus describing the Blocks World in which there is just a single action *move* defined by the applicability axiom Δ_a and effect axioms Δ_e:

$$\begin{gathered} applicable(move(x,y,z),s) \leftrightarrow clear(x,s) \wedge clear(z,s) \wedge on(x,y,s) \\ applicable(move(x,y,z),s) \rightarrow on(x,z,result(move(x,y,z),s)) \\ applicable(move(x,y,z),s) \rightarrow clear(y,result(move(x,y,z),s)) \end{gathered}$$

Note that there are no frame axioms Δ_f in this theory. Suppose the initial state depicted below is described by some formula Σ_{s_i}.



Give a logical formula F that does not follow from this theory due to the lack of frame axioms, i.e. give F such that:

$$\begin{gathered} \Sigma_{s_i} \wedge \Delta_a \wedge \Delta_e \not\models F, \text{ but} \\ \Sigma_{s_i} \wedge \Delta_a \wedge \Delta_e \wedge \Delta_f \models F. \end{gathered}$$

Answer:
For example: $on(B,A,result(move(C,B,E),s_i))$, i.e. B is still on A after moving C from B onto E.

(c) **State-Space Search**: The conceptual model underlying most planning approaches is the model of state transition systems. The state transition function γ for restricted state transition systems maps a given state s and action a to a new state. Extend this definition so that the second argument may be a plan π, i.e. define $\gamma(s,\pi)$.

Answer:
The extended state transition function for a plan $\pi = \langle a_1, \ldots, a_k \rangle$ is defined as follows:

$$\begin{array}{ll} \gamma(s,\pi) = s & \text{if } k = 0 \ (\pi \text{ is empty}) \\ \gamma(s,\pi) = \gamma(\gamma(s,a_1), \langle a_2, \ldots, a_k \rangle) & \text{if } k > 0 \text{ and } a_1 \text{ applicable in } s \\ \gamma(s,\pi) = undefined & \text{otherwise} \end{array}$$

(d) **Extended Representation**: In a STRIPS planning problem $P = (O, s_i, g)$ the goal g must be described by a set of ground literals. However, it is possible to allow for existentially quantified variables in goals and then rewrite the planning problem into an equivalent problem that contains only ground literals in goals. Show how this can be done for the following goal formula:

$$\exists x : on(x, c_1) \wedge colour(x, red)$$

Answer:
Rewrite the planning problem as $P' = (O', s_i, g')$, where:

- $g' = p$, where p is a new proposition symbol that does not occur in P; and
- $O' = O \cup (goalop(x), \{on(x, c_1), colour(x, red)\}, \{p\})$, where *goalop* is a new operator name in O.

In other words, we add a new operator that takes the existentially quantified variable as its parameter, has preconditions corresponding to the original goal, and the only effect is the rewritten goal g'.

(e) **HTN Planning**: An alternative view of planning is that we are not trying to achieve some explicit goals, but rather we want to perform some tasks in a given task network. The problem specification in this case includes some abstract tasks that need to be decomposed. This decomposition is performed using HTN methods. What are the components that make up an HTN method in an HTN planning domain?

Answer:
Let M_S be a set of method symbols. An HTN method is a 4-tuple $m = (name(m), task(m), subtasks(m), constr(m))$, where:

- $name(m)$, the name of the method, is a syntactic expression of the form $n(x_1, \ldots, x_k)$ in which
  - $n \in M_S$ is a unique method symbol and
  - $x_1, \ldots, x_k$ are all the variable symbols that occur in m;
- $task(m)$ is a non-primitive task; and
- the pair $(subtasks(m), constr(m))$ is a task network.

(f) **Representations**: Neoclassical planners like Graphplan appear to be limited by the fact that they only work on propositional domains. However, it can be shown that for every propositional planning problem P_P there is an equivalent ground STRIPS planning problem P_S and vice versa. Show how a propositional planning problem can be translated into a ground STRIPS planning problem.

Answer:
Let $P_P = (A, s_i, g)$ be a statement of a propositional planning problem. An equivalent ground STRIPS planning problem is given by $P_S = (A', s_i, g)$, where:

- $A' = \{(n', p', e') | a = (precond(a), effects^-(a), effects^+(a)) \in A$ and
  - n' is a unique operator name
  - $p' = precond(a)$ and
  - $e' = effects^+(a) \cup \{\neg p | p \in effects^-(a)\}\}$.

(g) **Scheduling**: Once a plan for a given planning problem has been found, what often remains to be done is the assignment of resources to the different actions in the plan. This problem is known as the scheduling problem and there are many

variants. For example, the following is an instance of the machine scheduling problem. What does this mean? Describe, in words, what the problem is.

- $j_1 : \langle r_1(3), r_2(3), r_1(2) \rangle$
- $j_2 : \langle r_2(3), r_1(5) \rangle$

Answer:
There are two jobs (sequences of actions) that require two different resource types. There may be multiple machines providing each resource type, but no information is given. The job requirements are:

- j_1 consists of three totally ordered actions
  - the first action requires 3 units of resource type r_1
  - the second action requires 3 units of resource type r_2
  - the third action requires 2 units of resource type r_1
- j_2 consists of two totally ordered actions
  - the first action requires 3 units of resource type r_2
  - the second action requires 5 units of resource type r_1

3. (a) **AI Planning**: The study of planning in Artificial Intelligence is closely linked with the goals of the field as a whole. Why should we study AI planning? What will we gain from this study?

   Answer:

   - for designing information processing tools that give access to affordable and efficient planning resources
   - planning is an important component of rational behaviour (scientific goal of AI)
   - as part of the study and engineering of autonomous intelligent machines (engineering goal of AI)

   (b) **Situation Calculus**: The first approach to planning was to use theorem provers to solve planning problems. For this, planning problems were written as first-order theories in the situation calculus. However, this gave rise to the frame problem, i.e. the need to explicitly represent frame axioms in the theory. What are the three principal ways in which the frame problem as it appears in the situation calculus has been addressed in AI planning?

   Answer:

   - use a different style of representation in first-order logic (same formalism)
   - use a different logical formalism, e.g. non-monotonic logic
   - write a procedure that generates the right conclusions and forget about the frame problem

   (c) **State-Space Search**: The planning problem can be seen as a search problem. In the state-space search approach, what is the search space? What do the nodes in this search space represent? What do the arcs represent? What does a path in this search space correspond to?

   Answer:

   - the search space is subset of state space
   - the nodes correspond to world states
   - the arcs correspond to state transitions
   - a path in the search space corresponds to plan

   (d) **Plan-Space Search**: An alternative to state-space search is plan-space search in which nodes in the search space are partial plans and arcs correspond to plan refinements. In plan-space search, what does the initial search state, i.e. the root node of the search tree, look like for a planning problem $P = (O, s_i, g)$?

   Answer:

   - represent initial state s_i and goal g as dummy actions
     - action *init*: no preconditions, initial state s_i as effects
     - action *goal*: goal conditions g as preconditions, no effects
   - initial search state: empty plan $\pi_0 = (\mathit{init}, \mathit{goal}, (\mathit{init} \prec \mathit{goal}), ,)$:
     - two dummy actions *init* and *goal*;
     - one ordering constraint: *init* before *goal*;
     - no variable bindings; and
     - no causal links.

(e) **HTN Extensions**: Practical planners often need to deal with domains in which numeric computations must be performed, e.g. adding weights or comparing numbers, or in which external data sources (databases) need to be queried. Defining this within a classical planning domain is not feasible and planners have been extended with a special mechanism to deal with such cases. What is this mechanism? How is it incorporated into a planning algorithm? What effect does this have on the formal properties that a planning algorithm might have without this extension?

Answer:

- attached procedures: associate predicates with procedures
- modify planning algorithm: evaluate preconditions by
  - calling the procedure attached to the predicate symbol if there is such a procedure
  - test against world state (set-relation, theorem prover) otherwise
- soundness and completeness: depends on procedures

(f) **Graphplan**: The planning graph developed by the Graphplan planner quickly grows from layer to layer until the proposition layers contain all those propositions that are eventually achievable. However, propositions that occur in the same layer may not be achievable simultaneously. More specifically, two mutually exclusive propositions cannot be achieved in the same proposition layer. Define this so-called mutex relation between propositions.

Answer:
Two propositions p and q in the jth proposition layer P_j are mutex (mutually exclusive) if:

- every action in the preceding action layer A_j that has p as a positive effect (incl. no-op actions) is mutex with every action in A_j that has q as a positive effect, and
- there is no single action in A_j that has both, p and q, as positive effects.

(g) **Temporal Planning**: One of the assumptions made for restricted state transition systems is that time is implicit. When temporal constraints are considered they are usually managed by a temporal constraint manager. Give an algorithm (in pseudo-code) that can be used to check that a temporal constraint network in the point algebra (PA) is consistent.

Answer:
In the point algebra the path consistency algorithm is complete, i.e. it can be used to check for global consistency of the network, not only path consistency. The algorithm for a given set of constraints C works as follows:

function $pathConsistency(C)$
  while $\neg C.isStable()$ do
    for each $k : 1 \leq k \leq n$ do
      for each pair $i, j : 1 \leq i < j \leq n, i \neq k, j \neq k$ do
        $c_{ij} \leftarrow c_{ij} \cap [c_{ik} \bullet c_{kj}]$
        if $c_{ij} = \emptyset$ then return inconsistent