Module Title: Automated Planning
Exam Diet (Dec/April/Aug): December 2004
Brief notes on answers:

1. (a) **AI Planning**: A number of restricting assumptions are often made in AI planning to simplify the problem. One of these assumptions is that the state-transition system Σ is *fully observable*. What does this assumption mean? In what type of domain do we want to drop this assumption? What issues will arise as a result? [*5 marks*]

   Answer:
   Fully observable Σ: the observation function η is the identity function.

   - Must be relaxed if: need to handle states in which not every aspect is or can be known.
   - Issues: if $\eta(s) = o$, $\eta^{-1}(o)$ usually contains more than one state (ambiguity) and the successor state is undefined.

   (b) **State-Space Search**: To show that an algorithm is sound and complete, it is necessary to formally define the problem that the algorithm solves and what constitutes a solution to such a problem. Give the formal definition of the planning problem for the STRIPS representation. Furthermore, define when a (total-order) plan constitutes a solution to a planning problem. [*5 marks*]

   Answer:
   A STRIPS planning problem is a triple $\mathcal{P} = (\Sigma, s_i, g)$ where:

   - $\Sigma = (S, A, \gamma)$ is a STRIPS planning domain on some first-order language $\mathcal{L}$
   - $s_i \in S$ is the initial state
   - g is a set of ground literals describing the goal such that the set of goal states is: $S_g = \{s \in S | s \textit{ satisfies } g\}$

   A plan π is a solution to a planning problem $\mathcal{P} = (\Sigma, s_i, g)$ if $\gamma(s_i, \pi)$ satisfies g.

   (c) **Plan-Space Search**: In plan-space search the nodes in the search space are partial plans that are refined until a solution plan is found. Partial plans that are not solutions may contain threats. Construct an example that explains the concept of a threat using the operators given in the planning domain defined in appendix 1. [*5 marks*]

   Answer:

   - partially instantiated actions in the plan:
     - `(op0 obj1 ?x2 ?x3)`
     - `(op1 ?x1 ?x2 obj1)`
     - `(op0 ?x1 ?x2 obj1)`
   - with `(op0 obj1 ?x2 ?x3)` before `(op1 ?x1 ?x2 obj1)`
   - and causal link `(op0 obj1 ?x2 ?x3)` $\overset{P1(obj1)}{\rightarrow}$ `(op1 ?x1 ?x2 obj1)`

Then the effect `(not (P1 obj1))` of `(op0 ?x1 ?x2 obj1)` constitutes a threat to the causal link. That is, if this operator was executed between the two connected by the causal link, it would undo the condition protected by the causal link.

(d) **Representation**: The STRIPS representation, the propositional representation, and the state-variable representation can be shown to be equally expressive. This is done by providing methods that can translate back and forth between the different representations. Translate the following operator in the state-variable representation into the STRIPS representation. How many operators in the propositional representation would be required to represent this operator?

- name: $svop(x, y, z)$
- precondition: $Sv1(x, y) = z$
- effects: $Sv1(x, y) \leftarrow y, Sv2(x) \leftarrow z$

[*5 marks*]

Answer:

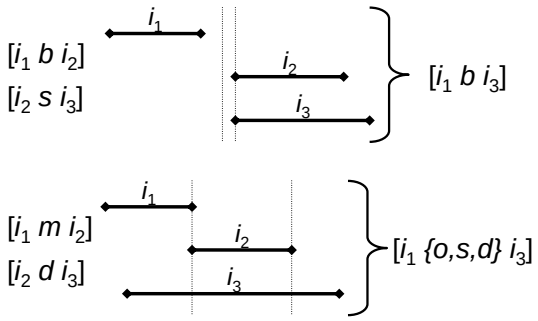
- name: $svop(x, y, z, v)$
- preconditions: $Sv1(x, y, z), Sv2(x, v)$
- effects: $Sv1(x, y, y), \neg Sv1(x, y, z), Sv2(x, z), \neg Sv2(x, v)$

Assume there are n objects in the domain. Then 4^n propositional operators are required.

(e) **Temporal Planning**: In the Interval Algebra there are 13 different primitive relations that can relate two intervals. These can be combined using the composition operator ($\bullet$). For example, $(b \bullet b)$ can be simplified to b. What relations are expressed by $(b \bullet s)$ and $(m \bullet d)$? Explain your answers. [*4 marks*]

Answer:

- $i_1(b \bullet s)i_3$ can be simplified to $i_1 b i_3$ (b = before, s = starts)
- $i_1(m \bullet d)i_3$ can be simplified to $i_1\{o, s, d\}i_3$ (m = meets, d = during, o = overlaps)



2. (a) **Situation Calculus**: In the situation calculus actions are denoted by function terms. The definition of these functions is given by a set of axioms, namely the applicability axioms, the effect axioms, and the frame axioms. Define these axioms for the action `op1` defined as part of the domain in appendix 1. [*5 marks*]

Answer:

- applicability:
  $\forall x, y, z, s : applicable(op1(x, y, z), s) \leftrightarrow P1(z, s) \wedge P0(x, y, s) \wedge \neg P0(z, x, s)$
- effect axiom:
  $\forall x, y, z, s : applicable(op1(x, y, z), s) \rightarrow P0(y, z, result(op1(x, y, z), s))$
- frame axiom:
  $\forall x, x', y, y', z, s : P0(x', y', s) \wedge (x' \neq z \vee y' \neq x) \rightarrow P0(x', y', result(op1(x, y, z), s))$
- frame axiom:
  $\forall x, x', y, y', z, s : \neg P0(x', y', s) \wedge (x' \neq x \vee y' \neq y) \rightarrow \neg P0(x', y', result(op1(x, y, z), s))$
- frame axiom:
  $\forall x, x', y, z, s : P1(x', s) \leftrightarrow P1(x', result(op1(x, y, z), s))$

(b) **State-Space Search**: Consider the planning domain and problem defined in appendix 1. The first step in the forward state-space search algorithm computes the set of applicable actions. What are the applicable actions for the given initial state? [*5 marks*]

Answer:

- `op0( obj2 obj0 obj1 )`
- `op0( obj2 obj2 obj1 )`
- `op1( obj2 obj1 obj1 )`
- `op1( obj0 obj1 obj1 )`
- `op1( obj2 obj1 obj0 )`
- `op1( obj0 obj1 obj0 )`

(c) **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 $P = (O, s_i, g)$? [*5 marks*]

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 $\pi_0 = (\{init, goal\}, (init \prec goal), \{\}, \{\})$:
  - two dummy actions *init* and *goal*;
  - one ordering constraint: *init* before *goal*;
  - no variable bindings; and
  - no causal links.

(d) **Task Networks**: When planning with task networks, the aim is not to achieve some goal state, but to accomplish a given set of tasks. Describe, in pseudo code, the ground partial-order forward decomposition algorithm (Ground-PFD) that takes an initial state s, an initial task network $w = (U, E)$, a set of primitive operators O, and a set of methods M. *[5 marks]*

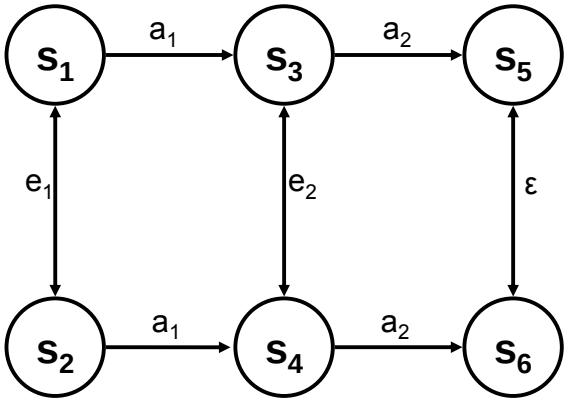
function Ground-PFD(s, w, O, M)
    if $w.U = \emptyset$ return $\langle\rangle$
    $task \leftarrow \{t \in U | t$ has no predecessors in $w.E\}$.chooseOne()
    if $task$.isPrimitive() then
        $actions \leftarrow \{(a, \sigma) | a = \sigma(task)$ and a applicable in $s\}$
        if $actions$.isEmpty() then return failure
        $(a, \sigma) \leftarrow actions$.chooseOne()
        $plan \leftarrow$ Ground-PFD$(\gamma(s, a), \sigma(w - \{task\}), O, M)$
        if $plan =$ failure then return failure
        else return $\langle a \rangle \bullet plan$
    else
        $methods \leftarrow \{(m, \sigma) | \sigma(m)$ is relevant for $task$ and m is applicable in $s\}$
        if $methods$.isEmpty() then return failure
        $(m, \sigma) \leftarrow methods$.chooseOne()
        return Ground-PFD$(s, \delta(w, task, m, \sigma), O, M)$

(e) **SAT Planning**: One approach to solving a bounded planning problem is to transform it into a satisfiability problem and use a SAT solver to find a plan. How are the actions in a planning domain encoded in this approach? When the SAT solver has finished, how can we extract the solution plan from the SAT solver's output? *[5 marks]*

Answer: Let A be the set of action propositions and $a \in A$.

- encoding actions:
  for $0 \leq i \leq n-1 : a_i \Rightarrow (\bigwedge_{f \in precond(a)} f_i \wedge \bigwedge_{f \in effects^+(a)} f_{i+1} \wedge \bigwedge_{f \in effects^-(a)} \neg f_{i+1})$
  where n is the size of the bounded problem
- if the SAT solver finds a_i to be true then a is the ith action in the plan

3. (a) **AI Planning**: The following graph represents a state-transition system $\Sigma = (S, A, E, \gamma)$ where a_1 and a_2 denote actions, e_1 and e_2 denote events, and ε denotes no event or action taking place. Define this state-transition system formally. Is this system deterministic? What state will the system be in after the action sequence $\langle a_1 a_2 \rangle$ assuming the initial state is s_1 and no events occur?



[*5 marks*]

Answer:

- $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$
- $A = \{a_1, a_2\}$
- $E = \{e_1, e_2\}$
-

γ	a_1	a_2	e_1	e_2	ε
s_1	$\{s_3\}$	$\{\}$	$\{s_2\}$	$\{\}$	$\{\}$
s_2	$\{s_4\}$	$\{\}$	$\{s_1\}$	$\{\}$	$\{\}$
s_3	$\{\}$	$\{s_5\}$	$\{\}$	$\{s_4\}$	$\{\}$
s_4	$\{\}$	$\{s_6\}$	$\{\}$	$\{s_3\}$	$\{\}$
s_5	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{s_6\}$
s_6	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{s_5\}$

The system is effectively non-deterministic because of the ε transition. $\langle a_1 a_2 \rangle$ will get the system into s_5 or s_6.

(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? [*5 marks*]

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**: Consider the planning domain and problem defined in appendix 1. The first step in the ground backward state-space search algorithm computes the set of relevant actions. What are the relevant actions for the given goal? [*5 marks*]

Answer:

- `op0( obj2 obj1 obj0 )`
- `op0( obj2 obj2 obj0 )`
- `op0( obj1 obj2 obj0 )`
- `op1( obj0 obj2 obj1 )`
- `op1( obj1 obj2 obj1 )`
- `op1( obj2 obj2 obj1 )`

(d) **Task Networks**: 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.

[*5 marks*]

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.

(e) **Graphplan**: The planning graph developed by Graphplan consists of action and proposition layers. Under what condition is a set of independent actions π (in one action layer) applicable to a state s? What is the result, $\gamma(s, \pi)$, of applying this set of independent actions? [*5 marks*]

Answer:

- A set π of independent actions is applicable in a state s if and only if $\bigcup_{a \in \pi} \text{precond}(a) \subseteq s$.
- The result of applying the set π in s is defined as: $\gamma(s, \pi) = (s - \text{effects}^-(\pi)) \cup \text{effects}^+(\pi)$, where:
  - $\text{effects}^+ = \bigcup_{a \in \pi} \text{effects}^+(a)$, and
  - $\text{effects}^- = \bigcup_{a \in \pi} \text{effects}^-(a)$.

Appendix 1

The following is definition of a planning domain and a statement of a planning problem in PDDL:

```
(define (domain random-domain)
  (:requirements :strips)

  (:action op0
    :parameters (?x1 ?x2 ?x3)
    :precondition (and
      (P1 ?x3) (P0 ?x1 ?x3) (P0 ?x2 ?x3)
      (not (P1 ?x1)))
    :effect (and
      (P1 ?x1) (P1 ?x2) (not (P1 ?x3))
      (not (P0 ?x1 ?x3)) (not (P0 ?x2 ?x3))))

  (:action op1
    :parameters (?x1 ?x2 ?x3)
    :precondition (and
      (P1 ?x3) (P0 ?x1 ?x2)
      (not (P0 ?x3 ?x1)))
    :effect (and (P0 ?x2 ?x3)))
)


(define (problem problem1)
  (:domain random-domain)

  (:init
   (P0 obj0 obj1)
   (P0 obj1 obj1)
   (P0 obj2 obj1)

   (P1 obj0)
   (P1 obj1))

  (:goal
    (and (P0 obj2 obj1) (P1 obj2)))
)
```