King's College London

This paper is part of an examination of the College counting towards the award of a degree. Examinations are governed by the College Regulations under the authority of the Academic Board.

Degree Programmes	BSc, MSci
Module Code	6CCS3AIP
Module Title	Artificial Intelligence Planning
Examination Period	January 2018 (Period 1)
Time Allowed	Two hours
Rubric	ANSWER THREE OF FOUR QUESTIONS.
	All questions carry equal marks. If more than three questions are answered, the three answers with highest marks will count.
Calculators	Calculators may be used. The following models are permitted: Casio fx83 / Casio fx85.
Notes	Books, notes or other written material may not be brought into this examination

PLEASE DO NOT REMOVE THIS PAPER FROM THE EXAMINATION ROOM

1.

a. Consider a temporal PDDL planning problem, which models a robot moving between two rooms. It is assumed that there is some remote control that the robot can press in order to open the door and that it will then remain open for 2 time units: notice that the open door action deletes its own precondition and can only ever be applied once.

Actions:

```
(:durative-action open-door
  :parameters (?d - door)
  :duration (= ?duration 2)
  :condition (and
      (at start (can-open ?d))
  )
  :effect (and
      (at start (not (can-open ?d)))
      (at start (open ?d))
      (at end (not (open ?d)))
))
(:durative-action move
  :parameters (?rob - robot ?from ?to - room ?d - door)
  :duration (= ?duration 8)
  :condition (and
      (at start (at ?rob ?from))
      (at end (open ?d))
      (over all (door-between ?from ?to ?d))
  )
  :effect (and
      (at start (not (at ?rob ?from)))
      (at end (at ?rob ?to))
))
```

Initial State: `(can-open door1) (at rob room1) (door-between room1 room2 door1)`
Goal:`(at robot room2)`

i. Show how a decision epoch planner would proceed to solve this problem from the initial state, by showing the states explored (including the propositions that are true, the timestamps of the states and the event queue). Can the planner solve the problem? If so, show how search proceeds to reach the goal; if not show the states that appear in search until the planner gets stuck and explain why the problem is not solvable from this point.

[6 marks]

ii. The search in CRIKEY3 generates plans that are propositionally sound, and then uses an STN to check whether the plan is temporally sound. Explain why the following plan is propositionally sound (despite not being temporally sound).


```
(open-door door1) start
(move rob room1 room2 door1) start
(move rob room1 room2 door1) end
(open-door door1) start
```


[3 marks]

iii. Draw the simple temporal network (STN) that CRIKEY 3 would produce for the plan in part ii (you may use the first letter of each action and the symbols $\vdash$ and $\dashv$ (for start and end) as abbreviations in your diagram if you wish).

[4 marks]

iv. Show how you can deduce from your STN that this plan is not temporally valid.

[2 marks]

b. This question is about preferences. Consider the following planning problem which models the daily activities of a cat.

```
(:action eat-tuna
  :parameters ()
  :precondition (and
      (been_cute)
      (in_house)
  )
  :effect (and
      (had_tuna)
      (had_dinner)
  )
)
```

```
(:action eat-cat-food
  :parameters ()
  :precondition (and
      (in_house)
  )
  :effect (and
    (had_dinner)
  ))
```



```
(:action take-nap
  :parameters ()
  :precondition (and
      (in_house)
  )
  :effect (and
      (slept)
))
```



```
(:action play-with-human
  :parameters ()
  :precondition (and
    (in_house)
  )
  :effect (and
    (been_cute)
  ))
```

Initial State: `(in_house)`

In an attempt to model the desires of a cat, that it must take a nap before dinner and a nap after dinner, a human has specified the following pair of preferences:

- (sometime-before had_dinner slept)
- (sometime-after had_dinner slept)

i. Sketch the automaton for the preference (sometime-after had-dinner slept)

[3 marks]

ii. Given the domain above, does a plan that satisfies both of these PDDL preferences necessarily satisty the cat's desire to nap both before and after dinner? Explain your answer.

[4 marks]

iii. Suppose the cat has 2 preferences: that it has had tuna, and that it only plays with the human (achieves been_cute) if it subsequently gets tuna. Write down PDDL preferences that would capture each of these requirements.

[3 marks]

2. This question is about PDDL+. For reference, here is *a fragment* of a variant of the *generator domain*:

Actions:

```
(:durative-action generate
 :parameters (?g - generator)
 :duration (>= ?duration 0)
 :condition ( and (over all (>= (fuelLevel ?g) 0))
                  (over all (safe ?g)))
 :effect (and (decrease (fuelLevel ?g) (* #t 1))
  (at start (increase (power-supplied) 10))
       (at end (decrease (power-supplied) 10))
)

(:action refuel
 :parameters (?g - generator ?t - tank)
 :precondition (and (not (using ?t ?g)) (available ?t))
 :effect (and (using ?t ?g) (not (available ?t)))
)

(:process refuelling
 :parameters (?g - generator ?t -tank)
 :precondition (and (using ?t ?g))
 :effect (and (increase (fuelLevel ?g) (* #t 2))
      (decrease (fuelInTank ?t) (* #t 1)))
)
```


The generator(s) must be used to provide enough power to satisfy the demand defined in the problem file. Each generator is initially filled to its 100-unit fuel capacity, and consumes fuel at a rate of 1 unit per second. When active, each generator provides 10 units of power. The generator can

be refilled using tanks, the action to do this is named refuel and increases the fuel level in the generator at a rate of 2 units per second.

a. In the fragment above, each generator can be used arbitrarily often. Instead, we want to change the model so that once a generator has been switched off it cannot be switched on again. Modify the domain to satisfy this new constraint.

[3 marks]

b. In the fragment above, once the process `refuelling` is triggered, it would run for ever. What is needed to avoid this odd behaviour and to have a correct domain for this problem?

[3 marks]

c. Extend the domain with the corresponding PDDL+ construct.

[4 marks]

d. In the real problem, the fuel level in the generator must not exceed the capacity of the generator. What is needed to guarantee this and to have a correct domain for this problem?

[4 marks]

e. Extend the domain with the corresponding PDDL+ construct.

[3 marks]

f. In the current model, it is possible to use more than one tank at the same time to refuel the generator. Describe how the domain needs to be modified so that only one tank at once can be used.

[2 marks]

g. Extend the domain so that only one tank at once can be used.

[2 marks]

h. In the current model, the requirement of producing enough power to satisfy the demand is not modelled. Assuming the function `demand` is used to represent the current demand, describe how the domain needs to be modified so that a valid plan will have to satisfy the demand requirements.

[4 marks]

3.

a. This part of the question is about planning with continuous numeric change in COLIN. Consider the domain below (which is unrelated to the one in question 1):

```
(:durative-action open-door
  :parameters ()
  :duration (= ?duration 2)
  :condition (and
      (at start (door-closed))
      (over all (>= (handle-angle) 30))
  )
  :effect (and
      (at start (not (door-closed)))
      (at end (door-open))))

(:durative-action turn-handle
  :parameters ()
  :duration (= ?duration 6)
  :condition (and
      (at start (handle-up))
      (over all (>= (handle-angle) 0))
  )
  :effect (and
      (at start (not (handle-up)))

      (at end (handle-up))
      (decrease (handle-angle) (* #t 10))
      (at start (increase (handle-angle) 60))))
```

Initial State `(handle-up) (= (handle-angle) 0) (door-closed)`
Goal: `(door-open)`

Write down the LP that COLIN would generate for the following plan:

- (turn-handle) start
- (open-door) start
- (open-door) end
- (turn-handle) end

[12 marks]

b. This part of the question is about landmarks.

i. What does it mean to say a fact or an action is a landmark for a given planning task?

[2 marks]

ii. Explain how planning can be done in stages through planning for each landmark. Why might this make planning more eficient, and in what situations might it make planning less efficient?

[3 marks]

c. This part of the question is about search.

i. One alternative to forward search planning is regression search, which instead searches backwards from the goal. What problem arises when performing regression search, due to the goal being a partial state, that often make it less efficient to solve problems in this manner?

[4 marks]

ii. How can restarts be helpful in local search? And why is randomness important in this setting?

[4 marks]

4. This is a question about SMTPlan, and encoding PDDL+ as SAT Modulo theories. Here is a fragment of a *Lander* domain:

Actions and Processes:

```
(:durative-action thrust
 :parameters (?l - lander)
 :duration (>= ?duration 0)
 :condition (over all (>= (height ?l) 0))
 :effect (and (decrease (v ?l) (* #t 15)))
)

(:action land
 :parameters (?l - lander)
 :precondition (= (height ?l) 0)
 :effect (and (landed ?l))
)

(:process descend
 :parameters (?l - lander)
 :precondition (>= (height ?l) 0)
 :effect (and
   (increase (v ?l) (* #t 9.8))
   (decrease (height ?l) (* #t (v ?l)))
 )
)
```


The lander must descend until the height is equal to zero, at which point it must land. As it descends, the lander is able to thrust in order to decrease the speed of its descent.

a. SMTPlan solves PDDL+ problems by encoding them as SMT formulae, and solving the formulae using an SMT solver. SMTPlan uses iterative deepening on the number of happenings.

i. What is meant by a happening in this context?

[1 marks]

ii. Explain how iterative deepening works in the context of SMTPlan. What effect does this approach have on finding optimal solutions to PDDL+ problems?

[3 marks]

iii. The zero-crossing problem occurs when dealing with a domain containing continuous non-linear change. Explain what is the zero-crossing problem, using this domain as an example.

[3 marks]

iv. Explain how the gradient of the continuous change can be used to alleviate the zero-crossing problem. In SMTPlan, what impact does this have on the kinds of dynamics that can be soundly solved?

[3 marks]

Here is a fragment of an SMT encoding of the lander problem:

```
(declare-fun (land lander01)0 () Bool)
(declare-fun (landed lander01)0_0 () Bool)
(declare-fun (landed lander01)0_1 () Bool)

(declare-fun (land lander01)1 () Bool)
(declare-fun (landed lander01)1_1 () Bool)
(declare-fun (landed lander01)1_0 () Bool)

(not (landed lander01)0_0))

(=> (landed lander01)0_1
    (or (landed lander01)0_0 (land lander01)0)))
(=> (not (landed lander01)0_1)
    (not (landed lander01)0_0)))
```

```
(=> (landed lander01)1_1
    (or (landed lander01)1_0 (land lander01)1)))
(=> (not (landed lander01)1_1)
    (not (landed lander01)1_0)))

(=> (land lander01)0 (landed lander01)0_1))
(=> (land lander01)1 (landed lander01)1_1))

(landed lander01)1_1)
```

There are four Boolean variables describing the *(landed lander01)* fact, just before and after two distinct happenings. There are two Boolean variables representing the application of the *(land lander01)* action. The constraints below show that *lander01* is not landed in the first happening, and that it should be in the second. There are also several implications.

b. Assuming that there are no other constraints containing the *(landed lander01)* variables, explain how the constraints above are not sufficient. Prove this by finding a valid assignment that does not correspond to the PDDL+ semantics.

[4 marks]

c. Extend the constraints with one or more new constraints to fix this issue.

[4 marks]

d. The following real variables are used to model the absolute time of the two happenings, the speed of the lander's descent, and the durative actions and processes during the interval between the happenings.

```
(declare-fun t0 () Real)
(declare-fun t1 () Real)

(declare-fun (v lander01)0_0 () Real)
(declare-fun (v lander01)0_1 () Real)
(declare-fun (v lander01)1_1 () Real)
(declare-fun (v lander01)1_0 () Real)

(declare-fun (descend lander01)0_run () Bool)
(declare-fun (thrust lander01)0_run () Bool)
```


Using these variables, extend the constraints above in order to:

i. Ensure that the initial happening occurs at time zero.

[1 marks]

ii. Ensure that the speed of the lander's descent does not change instantaneously within a happening.

[2 marks]

iii. Ensure a valid continuous change in *(v lander01)* between the happenings.

[4 marks]