ECE 479/579

Principles of Artificial Intelligence

Dr. Marefat

PROJECT # 2

“Planning”

Planning

In this assignment, you will interact with a planning system called UCPOP. The input to the planner consists of the problem statement and (since UCPOP is a *domain-independent* planner) the **domain theory** (or just **domain** for short). The problem statement is formed from the initial state of the world and the desired goal state, while the planning domain consists of a set of actions, each of which transforms the state of the world by instantaneously adding/deleting predicates to/from the world state.

This assignment consists of three parts. In part I, you are to encode the familiar blocksworld domain from the discussion of the STRIPS planner in class, and to run a few problems. In part II, you are to modify this domain to use fewer actions. In part III you are required to draw a planning graph. The syntax of problem and domain encodings are given later in this handout.

Part I: The blocksworld domain

1) Encode the four blocksworld actions discussed in class, namely, (pickup ?X), (putdown ?X), (stack ?X ?Y), and (unstack ?X ?Y). The following predicates are to be used, each having the same meaning as discussed in class:

(clear ?X)	(ontable ?X)
(on ?X ?Y)	(holding ?X)
(handempty).	

2) Use your encoding to solve the following two planning problems in UCPOP. (HINT: The planner will operate more efficiently if you encode the bottom-most goals first in the goal state.)

i) Problem name: **sussman-anomaly**

Initial State **Goal State**

ii) Problem name: **4BS1**

Initial State **Goal State**

Part II: Encoding the blocksworld domain using two actions

Assume the predicates (on ?X ?Y) and (ontable ?X) have been replaced by a single predicate, (onb ?X ?Y). For example, suppose we want to represent the fact that block A is on the table and block B is on top of block C. In part I, we would encode this as: (ontable A) and (on B C) respectively. In this part of the assignment, we can now encode these two facts as (onb A Table) and (onb B C) respectively.

1) In this part of the assignment, you are to encode the blocksworld domain using only two actions and using a subset of the predicates used in part I. The two actions can be named **puton** and **putontable.**

The predicates you may use in this part of the assignment are given below (HINT: you will only need a subset of these predicates):

(clear ?X)	(handempty)
(onb ?X ?Y)	(holding ?X)

Points will be given for the quality of action definitions (using minimum number of parameters and predicates).

2) Use your new two-action encoding to solve the sussman-anomaly problem and the 4BS1 problem in UCPOP as given above in Part I. Name these problems **sussman-anomaly-2** and **4BS1-2** respectively. In addition, also solve the following problem:

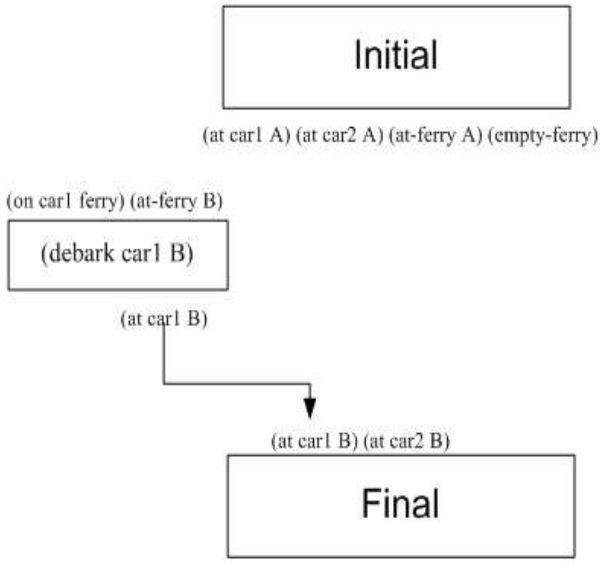
iii) Problem name: **4BSt**

Initial State **Goal State**

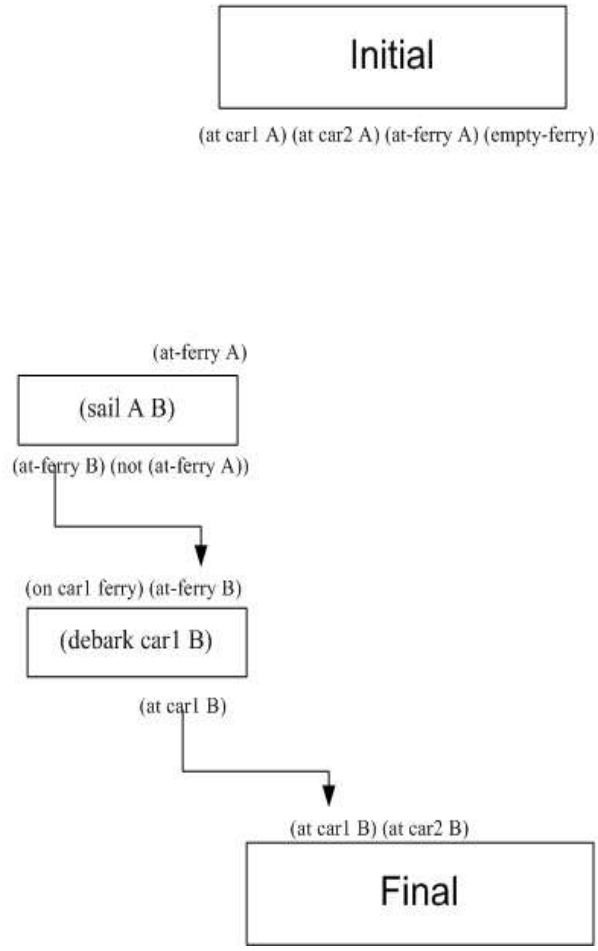
Part III: Drawing a planning graph

In this part you are required to draw and analyze the planning graph for the problem **4BS1** (using the domain from part I). The hard copy for this should include a pictorial description of the current plan status after each iteration of the planning algorithm. The plan status should show the actions, preconditions, causal links, open conditions etc. For example the following shows two successful iterations of the planning graph for the *Ferry* example:

STEP 1


Initial
(at car1 A) (at car2 A) (at-ferry A) (empty-ferry)
(on car1 ferry) (at-ferry B)
(debark car1 B)
(at car1 B)
(at car1 B) (at car2 B)
Final

STEP 2


Initial
(at car1 A) (at car2 A) (at-ferry A) (empty-ferry)
(at-ferry A)
(sail A B)
(at-ferry B) (not (at-ferry A))
(on car1 ferry) (at-ferry B)
(debark car1 B)
(at car1 B)
(at car1 B) (at car2 B)
Final

The following pages discuss the encoding of domains and problems in UCPOP and the `turnin` procedure for this assignment.

Specifications

Your program is required to adhere to the specifications listed here, otherwise it will not be graded.

PROGRAM NAME: the program name should be "mydomains.lsp" (all lowercase characters)

COMPILATION PROCEDURE: The following commands will be issued to compile your program (see page 14 for detailed instructions):

```
> (load “loader.lsp”)
> (load-ucpop-bin)
> (compile-file “mydomains.lsp”)
> (load "mydomains.fas")
```

Make sure your program will load with **no errors** using this procedure.

INPUT DESCRIPTION:

The first part of your “mydomains.lsp” file should contain a comment with the assignment number and your name, and the package identifier “(in-package “UCPOP”)”. For example,

```
;;; ECE 479 Project 2
;;; Joe Student
(in-package “UCPOP”)
```

The next part should contain your domain definitions, followed by your problem encodings.

We will use the ferry domain as an example input description. The ferry domain consists of two river banks (bank A and bank B), two cars (car1 and car2), and a ferry. The objective is to generate a plan that will use the ferry to transport the two cars between the river banks. The ferry may only transport one car at a time (see the figure on the next page). There are three actions (also called operators) in this domain:

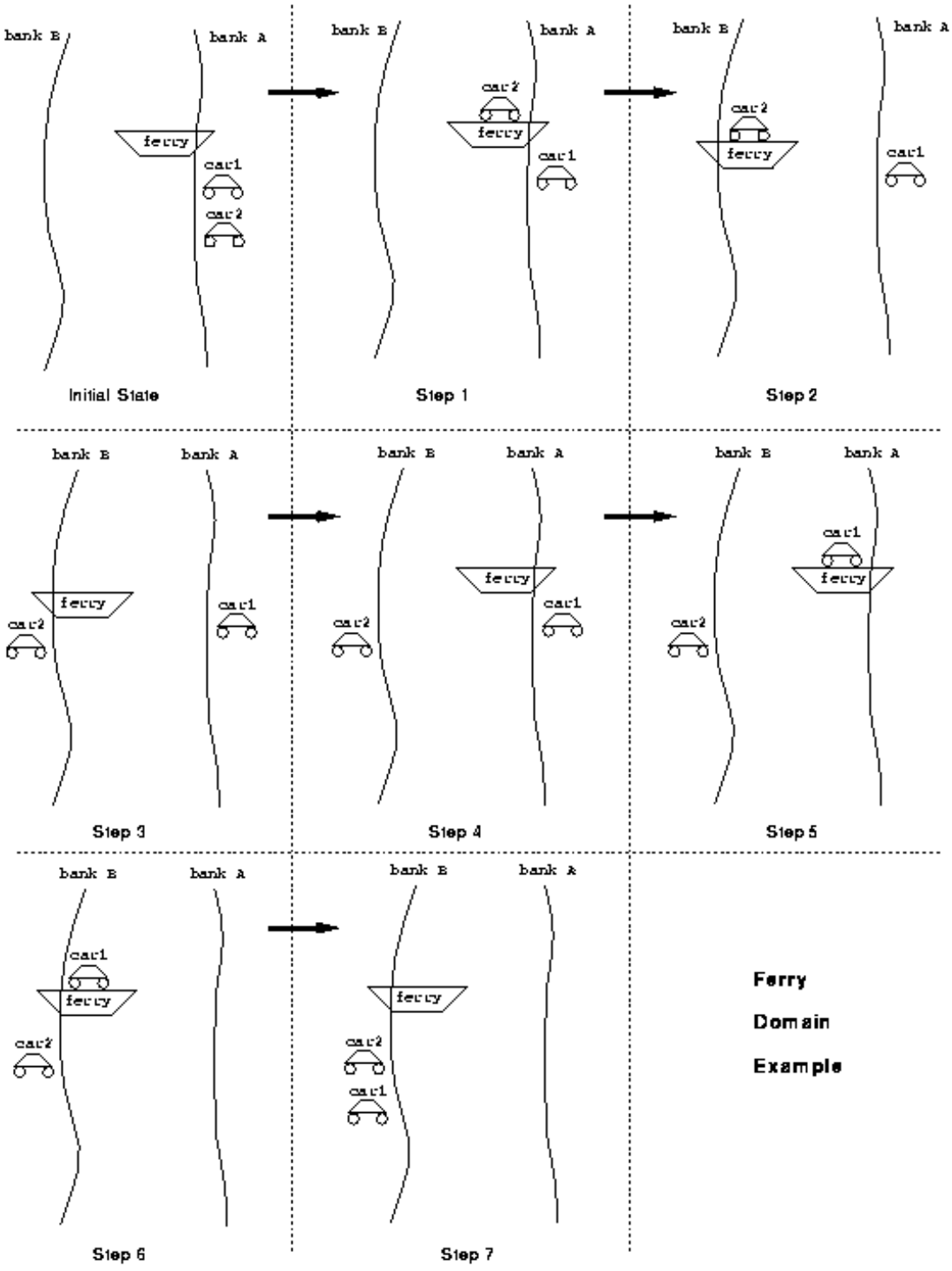
board - move a car from a bank onto the ferry
sail - move the ferry from one bank to the other
debark - move a car from the ferry to a bank

A problem in the ferry domain specifies the initial location of the ferry and two cars, and the final location of the two cars (we don’t care where the ferry ends up).

Domain Descriptions in UCPOP:

A domain definition in UCPOP contains a domain name followed by a set of operators:

```
(define (domain <domain name>)
        (:operator <op name>
                :parameters (?x ?y ... )
                :precondition (and (pred1 ...) (pred2 ...) ...)
                :effect (and (pred1 ...) (pred2 ...)
                        (not (pred3 ...)) (not (pred4 ...)))
        )
        (:operator <op name2>
                ...
        )  ...  )
```


bank B
bank A
ferry
car1
car2
Initial State
Step 1
Step 2
Step 3
Step 4
Step 5
Step 6
Step 7
Ferry
Domain
Example

<domain name> is the name of the domain. Let's call it "ferry-domain" in our example. Each operator definition consists of a name, a set of parameters, a set of preconditions, and a set of effects.

- <op name> is the name of the operator (board, sail, or debark in our example).
- The parameter set is simply the list of variables that are used in the preconditions and effects. These parameters are usually arranged in some order to provide meaning to the user.
Note that everywhere in UCPOP, variable names must be preceded by a question mark "?", and constant names must not begin with a question mark.
- The preconditions are those conditions that must be true before the operator can be applied (ie: the antecedent of the rule). If there is more than one precondition, then the preconditions must be placed in an **and** clause, as shown above.
- The effects set contains all the conditions which are added or deleted as a result of applying the operator (ie: the consequent of the rule). Like the preconditions, if there is more than one effect, the effects must be placed in an **and** clause. Also, to encode that a condition is deleted, it is placed in a **not** clause. For example, if an operator adds (sanded Block) and deletes (rough Block), this would be encoded as: (and (sanded Block) (not (rough Block))). sanded and rough are predicate names and Block is a constant.

We can now define our example ferry domain as follows:

```
(define (domain ferry-domain)          ; the name of this domain is:  ferry-domain

                                       ; the first operator is (board ?x ?y) which means that ?x
                                       ; boards the ferry at river bank ?y
  (:operator board                     ; define operator named: board
        :parameters (?x ?y)            ; declare the two parameters (they are both variables)
        :precondition (and (at ?x ?y) (at-ferry ?y) (empty-ferry))     ; declare the preconditons
        :effect                                                        ; declare the effects
        (and (on ?x ferry)                                             ; add effects
             (not (at ?x ?y))                                          ; delete effects
             (not (empty-ferry))))
                                       ; the second operator is (sail ?x ?y) which means that the
                                       ; ferry sails from bank ?x to bank ?y
(:operator sail                        ; define the operator named: sail
        :parameters (?x ?y)            ; declare the two parameters
        :precondition (at-ferry ?x)    ; declare the precondition
        :effect                                ; declare the effects
        (and (at-ferry ?y)                     ; add effects
             (not (at-ferry ?x))))             ; delete effects
                                       ; the third and final operator is (debark ?x ?y) which means
                                       ; ?x is moved from the ferry to bank ?y
  (:operator debark                    ; define the operator named: debark
        :parameters (?x ?y)            ; declare its two parameters
        :precondition (and (on ?x ferry) (at-ferry ?y))        ; declare the preconditions
        :effect                                ; declare the effects
        (and (at ?x ?y)                                ; add effects
              (empty-ferry)
```

```
        (not (on ?x ferry))))          ; delete effects
)                                      ; end of domain definition
```

Since you will be creating two domains (one for each part of this assignment), you will need two domain definitions. The name of your domains should be "blocks-world" for part I and "my-blocks-world" for part II; therefore, your first domain definition should start with:

```
(define (domain blocks-world)
```

and your second domain definition should start with:

```
(define (domain my-blocks-world)
```

The final part of your "mydomains.lsp" file should contain your problem definitions.

Problem Definitions in UCPOP:

A problem definition in UCPOP is of the form:

```
(define (problem <prob-name>)
        :domain '<domain-name>
        :inits ( (pred1 ...) (pred2 ...) ... )
        :goal (and (pred3 ...) (pred4 ...) ...)
)
```

- <prob-name> is the name of the problem. In our example, let's use the name "ferry-run".
- <domain-name> is the name of the domain in which the problem is to run. From the domain definition above, the domain name for our example is "ferry-domain".
- The inits are the initial conditions (or initial state) of the problem. Since the initial conditions form an implicit conjunction in UCPOP, you should **not** place the initial conditions in an **and** clause.
- The goal is the goal state for the problem. Here you must use an **and** clause to encode multiple goal conditions.

In our ferry example, let's assume that we initially have both cars and the ferry on bank A (as shown in the upper-left cornet of the figure), and our goal is to have both cars on bank B (as shown in the lower-middle of the figure). We will call this problem, "ferry-run". We can encode this problem as follows:

```
(define (problem ferry-run)                  ; define the problem: ferry-run
   :domain 'ferry-domain                     ; use the domain: ferry-domain
   :inits ((at car1 A) (at car2 A) (at-ferry A)   ; initial state
        (empty-ferry))
   :goal (and (at car1 B) (at car2 B))       ; goal state
)                                            ; end of problem definition
```

For your assignment, you are to use the names given with the drawings in part I and part II as the problem names.

RUNNING UCPOP:

Once you have loaded UCPOP and compiled and loaded your "mydomains.lsp" file as discussed above in the COMPILATION PROCEDURE, you are ready to run your problems. The first thing you must do is make sure you are in the UCPOP package. Type (in-package "UCPOP") at the lisp prompt.

To run a problem, use the "bf-control" function with the quoted problem name as an argument, which will initiate a best-first search to solve the problem. For example, to run the "ferry-run" problem in our example ferry domain, one should type: (bf-control 'ferry-run) at the lisp prompt.
The output of UCPOP will contain some statistics and the solution plan, if one was found. The output of UCPOP for our example "ferry-run" problem is shown below:

```
> (bf-control 'ferry-run)

Initial  : ((AT CAR1 A) (AT CAR2 A) (AT-FERRY A) (EMPTY-FERRY))

Step 1  : (BOARD CAR1 A)        Created 2
        0  -> (EMPTY-FERRY)
        0  -> (AT-FERRY A)
        0  -> (AT CAR1 A)
Step 2  : (SAIL A B)            Created 3
        0  -> (AT-FERRY A)
Step 3  : (DEBARK CAR1 B)       Created 1
        3  -> (AT-FERRY B)
        2  -> (ON CAR1 FERRY)
Step 4  : (SAIL B A)            Created 6
        3  -> (AT-FERRY B)
Step 5  : (BOARD CAR2 A)        Created 5
        1  -> (EMPTY-FERRY)
        6  -> (AT-FERRY A)
        0  -> (AT CAR2 A)
Step 6  : (SAIL A B)            Created 7
        6  -> (AT-FERRY A)
Step 7  : (DEBARK CAR2 B)       Created 4
        7  -> (AT-FERRY B)
        5  -> (ON CAR2 FERRY)

Goal    : (AND (AT CAR1 B) (AT CAR2 B))
        4  -> (AT CAR2 B)
        1  -> (AT CAR1 B)
Facts:
Complete!

UCPOP Stats: Initial terms = 4 ;   Goals = 3 ;  Success (7 steps)
     Created 1264 plans, but explored only 755
     CPU time:    4.1700 sec
     Branching factor:  1.245
```

Working Unifies: 3419
Bindings Added: 539
#plan<S=8; O=0; U=0; F=0>
#Stats:<cpu time = 4.1700>

Following the steps in numerical order (also see the figure), we see that the solution to our example problem is:

Step1: (board car1 A)
Step2: (sail A B)
Step3: (debark car1 B)
Step4: (sail B A)
Step5: (board car2 A)
Step6: (sail A B)
Step7: (debark car2 B)

TURNIN PROCEDURE:

To submit your assignment, you will need to use the `turnin` program as usual. **The hard copy (planning graph) needs to be turned in before the deadline in person.**

As previously discussed, you will be submitting a single text file called "mydomains.lsp". Your turnin should be typed like:

```
turnin ece479 proj2 mydomains.lsp
```

Your file will be submitted and you will be mailed a confirmation message shortly after submission.

Notes: (PAY CLOSE ATTENTION TO THESE!)

For this assignment, make sure you only submit a single text file.
This file should contain both the domains and all problem encodings.

Note that the name of the file is "mydomains.lsp" (all lower-case characters).

Make sure the file you turn in is properly formatted in the unix text format (ie: not full of MSDOS carriage returns that show up in the source as ^M when using vi). You can use the `dos2unix` command to help you with this.

It is YOUR RESPONSIBILITY to ensure that your facts and rules run properly on the shell machine using CLISP. Be sure to make the appropriate modifications if you do your development on other machines.

Instructions for loading and using UCPOP

(only the contents within ‘’ need to be typed in!)

Loading UCPOP

1) Copy the loader to a local directory (don’t type the single quotes):
   'cp /home/ece479/ucpop-bin/loader.lsp .'

2) Invoke lisp
   'clisp'

3) Load the ucpop loader
   '(load "loader.lsp")'

4) Run the binary loader
   '(load-ucpop-bin)'

Using the domains and problems you've created

Create new domains in your local directory. Let's say they are called 'mydomains.lsp'. These can be compiled as follows:

‘(compile-file "mydomains.lsp")’

After compiling a domain, you can load it using the "load" command: (after ucpop is loaded):

‘(load "mydomains.fas")’

Each time you change a domain file, you must recompile it and load it.