**PART 1:**

1)

Tower of Hanoi Problem:-

Init( peg(a) ^ peg(b) ^ peg(c) ^ on(d1,d2) ^ on(d2,a) ^ bigger(d2,d1)

clear(b) ^ clear(c) )

Goal(on(d1,d2) ^ on(d2,c)).

Action(

moveToDisk(disk,source,destiDisk),

PreCond: clear(destiDisk),clear(disk),on(disk,source),bigger(destDisk,disk)

Effect: ~clear(destiDisk), on(disk, destiDisk),clear(source),~on(disk,source)

)

Action(

moveToPeg(disk,source,destiPeg),

PreCond: clear(disk),on(disk,source),clear(destiPeg),peg(destiPeg)

Effect: ~clear(destiPeg), on(disk, destiPeg),clear(source), ~on(disk,source)

)

I have used symbol '~' as to denote negation

2)

1-

2- doing regression from sub-goals of final goal. trying to achieve each subgoal independetly.

3-

4- Final POP

3)since init is empty & this is STRIPS, it means negation of every condition or literal by closed world principle. My formulation is Inspired from problem formulation in the book.

Init( ), Goal( LeftShoeOn ^ RightShoeOn ^ HatOn ^ CoatOn )

Action( LeftSock

PRECOND :

EFFECT : LeftSockOn ) ,

Action( RightSock

PRECOND :

EFFECT : RightSockOn ),

Action( LeftShoe

PRECOND : LeftSockOn

EFFECT : LeftShoeOn ),

Action( RightShoe

PRECOND : RightSockOn

EFFECT : RightShoeOn )

Action( PutHatOn

PRECOND :

EFFECT : HatOn ),

Action( PutCoatOn

PRECOND :

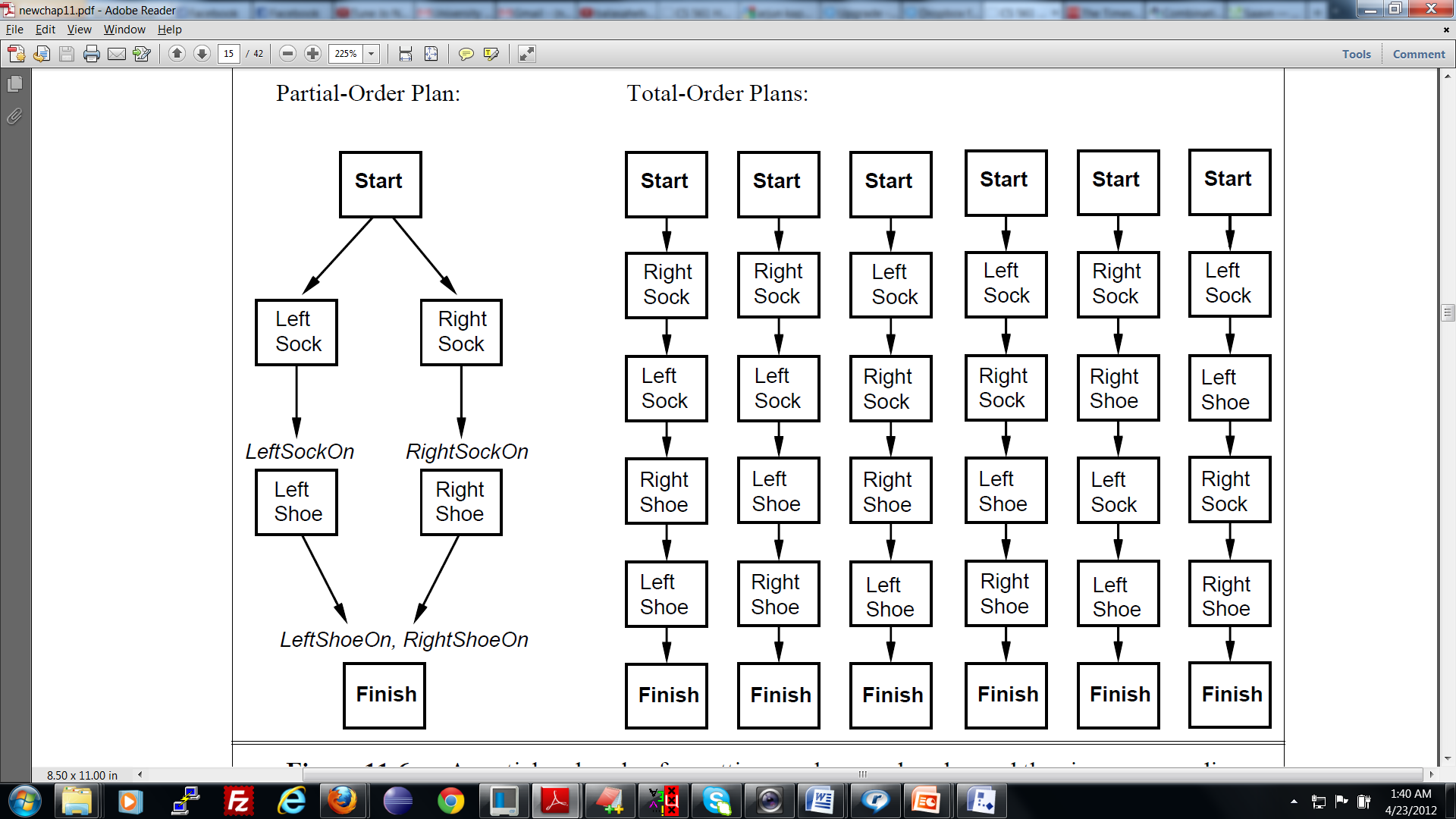
EFFECT : CoatOn ),

**3-1**



**3-2)**

**Partial Order Plan**



For Shoe problem we know from the plan that, there are 6 linearization or total order plans each having 4 actions.

Hence we can have any one of

PutHatOn OR PutCoatOn

be placed anywhere before or after these 4 actions giving us 5 combination for each of these 6 plans. So total number of possible plans is 6 \* 5=30

Now for each of these 30 plans, each having 5 actions in all(together with this new action), remaining action of PutHatOn OR PutCoatOn can be done in 6 ways. So total count by simple combinotorics is 30 \* 6 = 180

3-3

What is the minimum number of different planning graph solutions needed to represent all 180 linearizations?

ANSWER: ***1***

## PART 2: Natural Language

## 1: PP-attachment ambiguity

## parse2(s,[the,man,eats,the,sushi,with,the,tuna]).

## s(

## np(

## det(the)

## n(man))

## vp(

## vt(eats)

## np(

## np(

## det(the)

## n(sushi))

## pp(

## p(with)

## np(

## det(the)

## n(tuna))))))

## true ;

## s(

## np(

## det(the)

## n(man))

## vp(

## vp(

## vt(eats)

## np(

## det(the)

## n(sushi)))

## pp(

## p(with)

## np(

## det(the)

## n(tuna)))))

## true ;

## false.

**2: relative-clause attachment ambiguity.**

***parse2(s,[the,man,eats,the,sushi,with,the,woman,who,likes,the,tuna]).***

***This generates 3 parse trees,& displayes it on the screen.***

**parse2(s,[the,man,eats,the,sushi,with,the,woman,who,eats,the,sushi,with,the,tuna]).**

**this generates 13 trees.**

**3: pretty format: First two inputs for 1 & 2 are already printed in pretty format**

**4:For original parse that is parse2 in my case , breaks given string 2 substrings(whose concatenation gives original string) in all possibilities & does this recursively. Also it applies every production rule to all of these strings. This is because of lack of memoisation found in dynamic programming.**

**moreover it solves same sub problems again & again. Hence it is exponential.**





**5-**

**for the input of**

**parse3(s,[the,man,eats,the,sushi,with,the,woman,who,eats,the,sushi,with,the,tuna]).**

**it shows signifcant improvement over parse2. for producing final false parse2 waits for 10 seconds while parse3 produces final false within less than a second.**

improved complexity of parse3 is O(n^3 \* G).n is number of words in a string & G size of CNF grammar. It employs bottom-up parsing and dynamic programming.

**I am memoizing both if a string can or cannot be parsed by some production rule. So same recomputation is totally avoided.Two predicates asserted for this are already\_parsed\_successfully & cantbe\_parsed\_successfully respectively.**

**6:**

**parse4(s,[the,man,eats,the,sushi,with,the,tuna],P).**

**s(**

**np(**

**det(the)**

**n(man))**

**vp(**

**vt(eats)**

**np(**

**np(**

**det(the)**

**n(sushi))**

**pp(**

**p(with)**

**np(**

**det(the)**

**n(tuna))))))**

**P = 5.90625e-7 ;**

**false.**

**one can also try, parse4(s,[the,man,eats,the,sushi,with,the,woman,who,eats,the,sushi,with,the,tuna],P).**

**it gives two trees with same probability. Even parse4 employs memoisation.**

**parse4(s,[the,man,eats,the,sushi,with,the,woman,who,eats,the,sushi,with,the,tuna],P).**