Exercise Set 2

PDDL

- Create a PDDL problem where:
  1. There is *at least* 6 atomic fluents, 4 actions, and 2 paths to a goal state.
  2. All paths to the goal correspond to actions that can be relaxed to a *partial order*.

In the following solution, one of the paths to goal cannot be relaxed to a partial order in the sense desired here (it always remains a total order). But there is such a path that will be used later.

InitialState: -

Goal: Can-Fly

Action: CatchChicken

- Preconditions: -Tired
- Effect: HaveChicken, Tired

Action: BuyTicket

- Preconditions: HaveMoney
- Effect: CanFly,-HaveMoney

Action: GlueFeathersOn

- Preconditions: HaveFeathers, HaveGlue
- Effect: CanFly, -HaveFeathers,-HaveGlue

Action: PluckChicken

- Preconditions: HaveChicken
- Effect: HaveFeathers, -HaveChicken

Action: Work

- Preconditions: -Tired
- Effect: HaveMoney, Tired

Action: BuyGlue

- Precondition: HaveMoney
- Effect: HaveGlue, -HaveMoney

Action: Rest

- Precondition: Tired
- Effect: -Tired

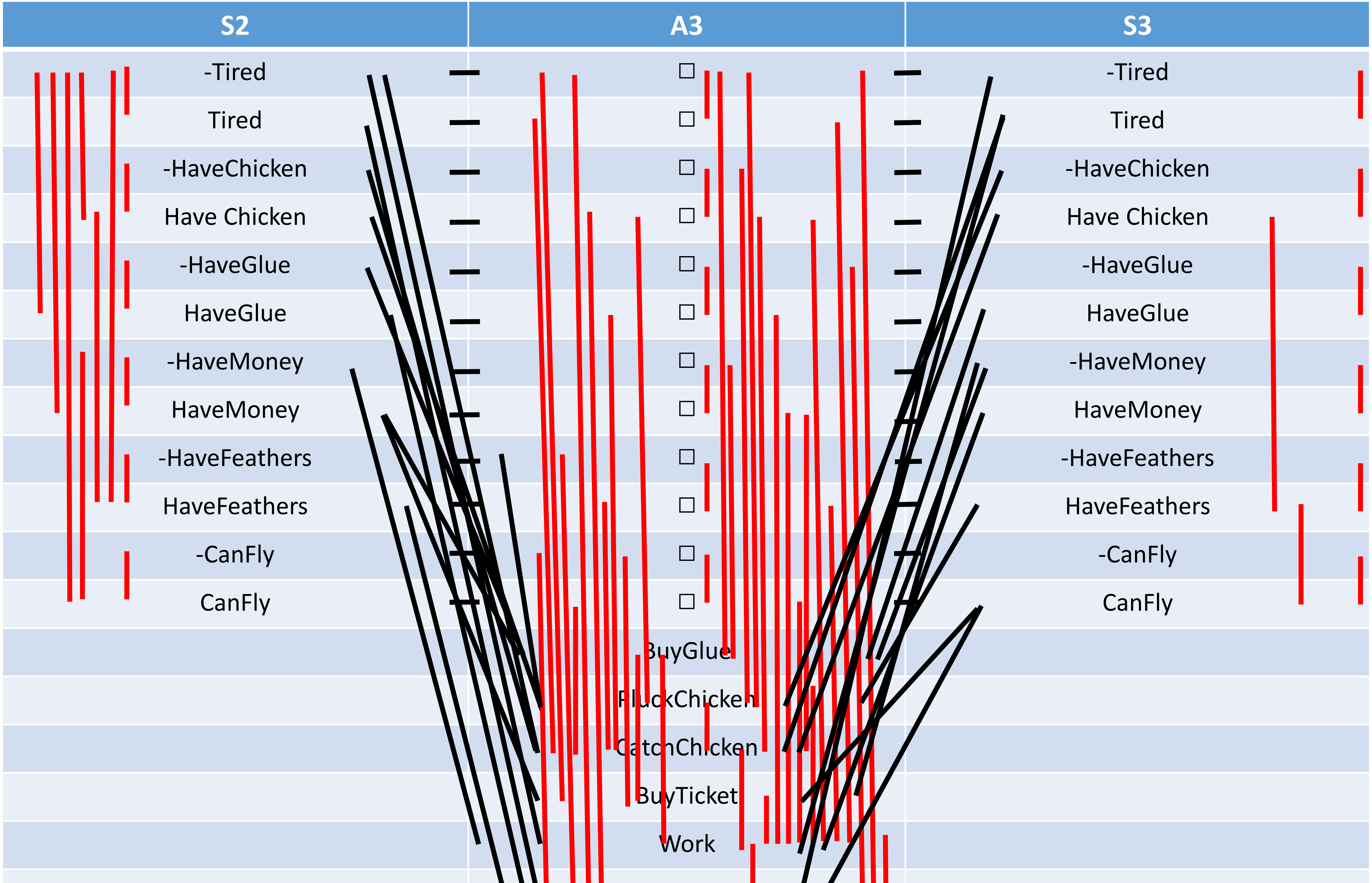
Planning Graphs

- Create the layers S_0 to S_3 (and therefore including S_0-A_2) of the planning graph for your problem, where S_0 is your specified initial state.

The answer here get more approximate as it goes! Planning Graphs are very hard to do by hand. You will not have to do anything this hard in the exam.

S0	A1	S1
-Tired	☐	-Tired
		Tired
-HaveChicken	☐	-HaveChicken
	CatchChicken	Have Chicken
-HaveGlue	☐	-HaveGlue
-HaveMoney	☐	-HaveMoney
	Work	HaveMoney
-HaveFeathers	☐	-HaveFeathers
-CanFly	☐	-CanFly

S1	A2	S2
-Tired	□	-Tired
Tired	□	Tired
-HaveChicken	□	-HaveChicken
Have Chicken	□	Have Chicken
-HaveGlue	□	-HaveGlue
	CatchChicken	HaveGlue
-HaveMoney	□	-HaveMoney
HaveMoney	□	HaveMoney
-HaveFeathers	□	-HaveFeathers
	Work	HaveFeathers
-CanFly	□	-CanFly
	BuyTicket	CanFly
	BuyGlue	
	PluckChicken	
	Rest	

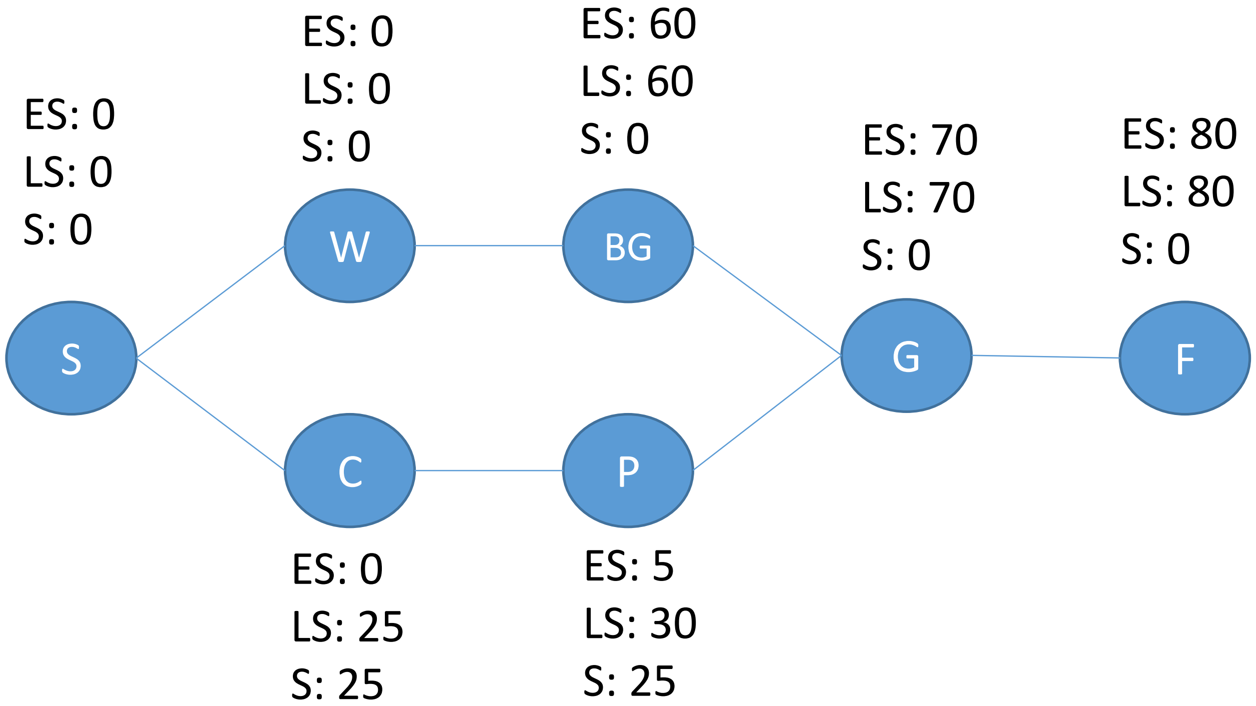

S2
A3
S3
-Tired
Tired
-HaveChicken
Have Chicken
-HaveGlue
HaveGlue
-HaveMoney
HaveMoney
-HaveFeathers
HaveFeathers
-CanFly
CanFly
BuyGlue
PluckChicken
CatchChicken
BuyTicket
Work
-Tired
Tired
-HaveChicken
Have Chicken
-HaveGlue
HaveGlue
-HaveMoney
HaveMoney
-HaveFeathers
HaveFeathers
-CanFly
CanFly

S0	A1	S1		S2		S3
-Tired	□	-Tired	□	-Tired		-Tired
		Tired	□	Tired		Tired
-HaveChicken	□	-HaveChicken	□	-HaveChicken		-HaveChicken
	CatchChicken	Have Chicken	CatchChicken	Have Chicken		Have Chicken
-HaveGlue	□	-HaveGlue	□	-HaveGlue		-HaveGlue
			BuyGlue	HaveGlue		HaveGlue
-HaveMoney	□	-HaveMoney	□	-HaveMoney		-HaveMoney
	Work	HaveMoney	Work	HaveMoney		HaveMoney
-HaveFeathers	□	-HaveFeathers	□	-HaveFeathers		-HaveFeathers
			PluckChicken	HaveFeathers		HaveFeathers
-CanFly	□	-CanFly	□	-CanFly		-CanFly
			BuyTicket	CanFly		CanFly

Resource Free Scheduling

- Add duration specifications to your actions.
- Take a partial order plan for your problem and use the Dynamic Programming Algorithm for Resource-Free Scheduling to calculate a schedule: Earliest start times, latest start times, and slack times for each required action.

	Action	Duration
BG	BuyGlue	10
P	PluckChicken	40
C	CatchChicken	5
BT	BuyTicket	15
W	Work	60
R	Rest	60
G	GlueFeathersOn	10

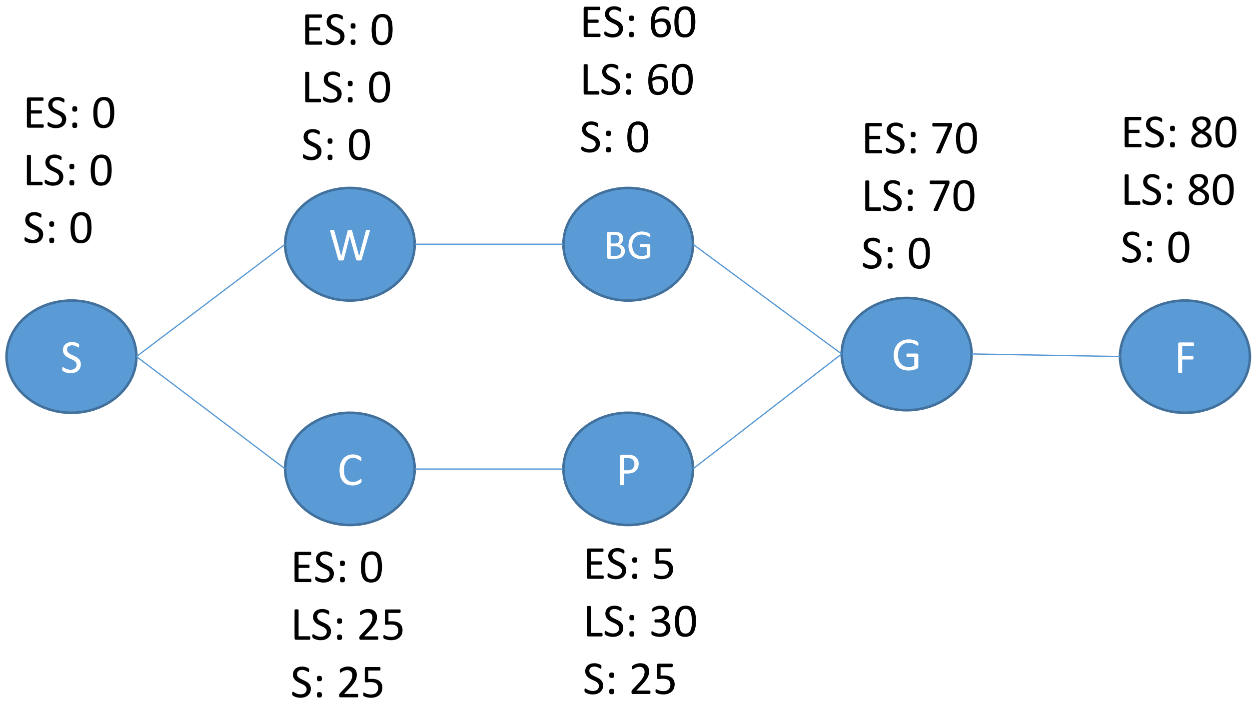


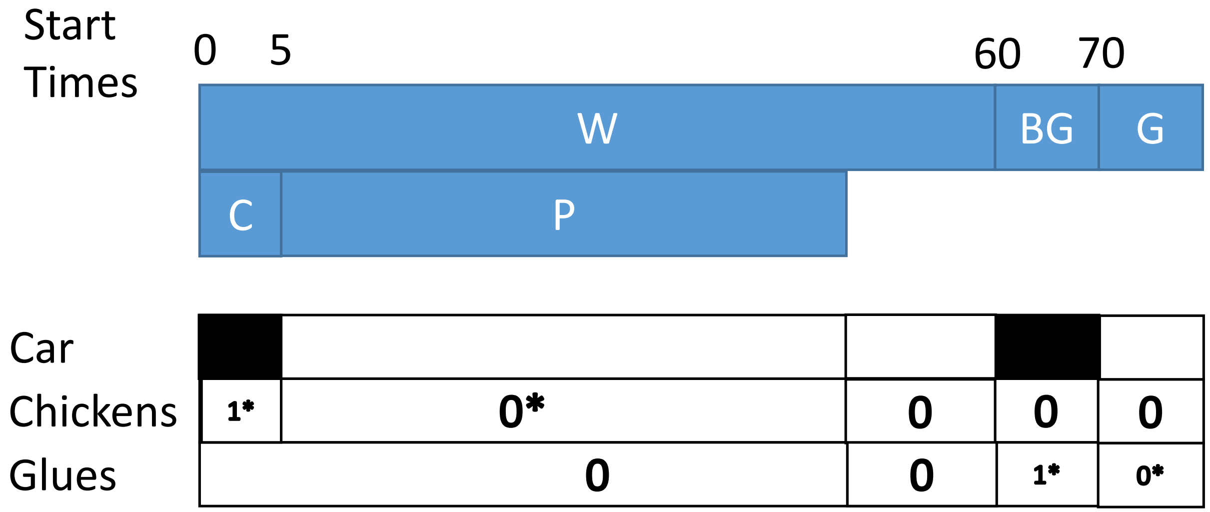
Resource Constrained Scheduling

- Add consumable and durable resource specifications to your actions.
- Use the minimum slack algorithm to calculate a resource constrained schedule: Start times and resource allocations.

There was a mistake on this question: It should be asking for start time and resource allocations only.

	Action	Duration	Uses	Consumes
BG	BuyGlue	10	Car	-1 glue
P	PluckChicken	40		1 chicken
C	CatchChicken	5	Car	-1 chicken
BT	BuyTicket	15	Car	
W	Work	60		
R	Rest	60		
G	GlueFeathersOn	10		1 glue





*The comsumable resource is *becoming* this number.