CPSC 322
Introduction to Artificial Intelligence

November 17, 2004

Things...

Term project is due in less than two weeks from today

The final exam will be at noon on Friday, December 10, in MCML 166

Move generation for the peg puzzle

A simple observation: if we represent the state of a peg puzzle like this:

```
[b,-,b,-,r,r]
```

then generating a new state that results from moving a blue peg one space to the right is nothing more than replacing a subsequence like this `[b,-]` with a subsequence like this `[-,b]`. So by scooting along the list above and collecting a new state every time we find a `[b,-]` and replace it with `[-,b]`, we get this collection of new states:

```
[[-,b,b,-,r,r],[b,-,-,b,r,r]]
```

Move generation for the peg puzzle

Similarly, a blue peg jumping over a red peg to the right is just this replacement:

`[b,r,-]` is replaced by `[-,r,b]`

Likewise, a red peg moving one space to the left and a red peg jumping over a blue peg to the left are these replacements, respectively:

`[-,r]` is replaced by `[r,-]`
`[-,b,r]` is replaced by `[r,b,-]`

Move generation for the peg puzzle

So if we can construct a procedure to scoot along a list from left to right and replace one occurrence of a given subsequence with another desired subsequence, AND if we can collect all the different lists that result from making those replacements...

...then we have the basis for a move generator for our beloved peg puzzle.

Move generation for the peg puzzle

```
/* pegpuzzle2.ci */

gen_all_moves(X,Movelist) <- gen_all_blue_moves(X,Bluemoves) &
                             gen_all_red_moves(X,Redmoves) &
                             append(Bluemoves,Redmoves,Movelist).

gen_all_blue_moves(X,Bluemoves) <- gen_all_blue_slides(X,Blueslides) &
                                   gen_all_blue_jumps(X,Bluejumps) &
                                   append(Blueslides,Bluejumps,Bluemoves).

gen_all_red_moves(X,Redmoves) <- gen_all_red_slides(X,Redslides) &
                                 gen_all_red_jumps(X,Redjumps) &
                                 append(Redslides,Redjumps,Redmoves).

gen_all_blue_slides(Board,Newboards) <-
     all_replacements(Board,[b,-],[-,b],Newboards).

gen_all_blue_jumps(Board,Newboards) <-
     all_replacements(Board,[b,r,-],[-,r,b],Newboards).

gen_all_red_slides(Board,Newboards) <-
     all_replacements(Board,[-,r],[r,-],Newboards).

gen_all_red_jumps(Board,Newboards) <-
     all_replacements(Board,[-,b,r],[r,b,-],Newboards).
```

Move generation for the peg puzzle

```
all_replacements(Board,Oldseq,Newseq,[Newhead|Newrest]) <-
          replace_subseq(Board,Oldseq,Newseq,Newhead) &
          all_replacements(Board,Oldseq,Newseq,Newrest) &
          no_duplicates([Newhead|Newrest]).

all_replacements(Board,Oldseq,Newseq,[]).

replace_subseq(Oldseq,Oldsubseq,Newsubseq,Newseq) <- append(L1,L2,Oldseq) &
                                                     append(Oldsubseq,L3,L2) &
                                                     append(Newsubseq,L3,L4) &
                                                     append(L1,L4,Newseq).
```

Move generation for the peg puzzle

```
append([],Z,Z).
append([A|X],Y,[A|Z]) <- append(X,Y,Z).

no_duplicates([]).
no_duplicates([H|T]) <- notmember(H,T) &
                        no_duplicates(T).

notmember(X,Y) <- ~member(X,Y).

member(H,[H|T]).
member(X,[H|T]) <- member(X,T).

/*
notmember(X,[]).
notmember(X,[H|T]) <- X\=H & notmember(X,T).
*/
```

Move generation in Prolog

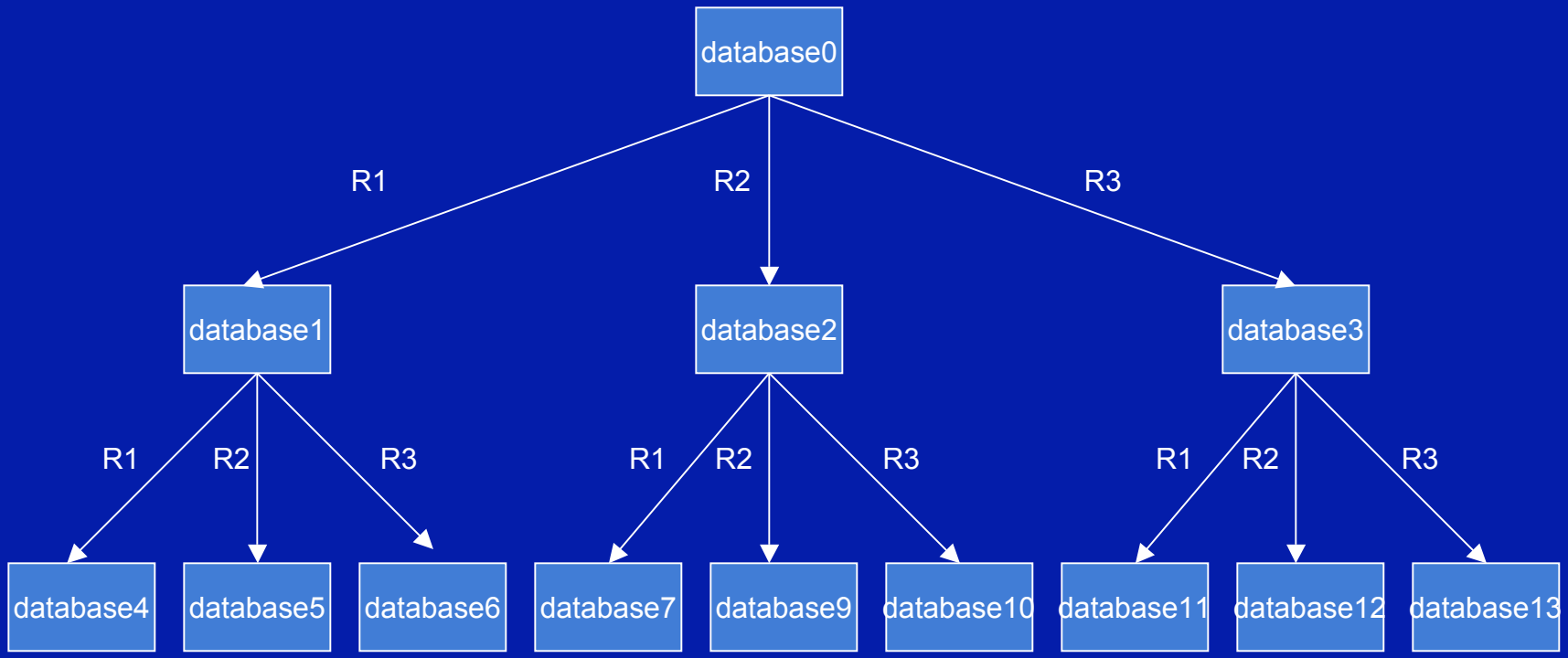
```
/* pegpuzzle.pl */

gen_all_moves(X,Movelist) :- gen_all_blue_moves(X,Bluemoves) ,
                             gen_all_red_moves(X,Redmoves) ,
                             append(Bluemoves,Redmoves,Movelist).

gen_all_blue_moves(X,Bluemoves) :- gen_all_blue_slides(X,Blueslides) ,
                                   gen_all_blue_jumps(X,Bluejumps) ,
                                   append(Blueslides,Bluejumps,Bluemoves).

gen_all_red_moves(X,Redmoves) :- gen_all_red_slides(X,Redslides) ,
                                 gen_all_red_jumps(X,Redjumps) ,
                                 append(Redslides,Redjumps,Redmoves).

gen_all_blue_slides(Board,Newboards) :-
     all_replacements(Board,[b,-],[-,b],Newboards).

gen_all_blue_jumps(Board,Newboards) :-
     all_replacements(Board,[b,r,-],[-,r,b],Newboards).

gen_all_red_slides(Board,Newboards) :-
     all_replacements(Board,[-,r],[r,-],Newboards).

gen_all_red_jumps(Board,Newboards) :-
     all_replacements(Board,[-,b,r],[r,b,-],Newboards).
```

Move generation in Prolog

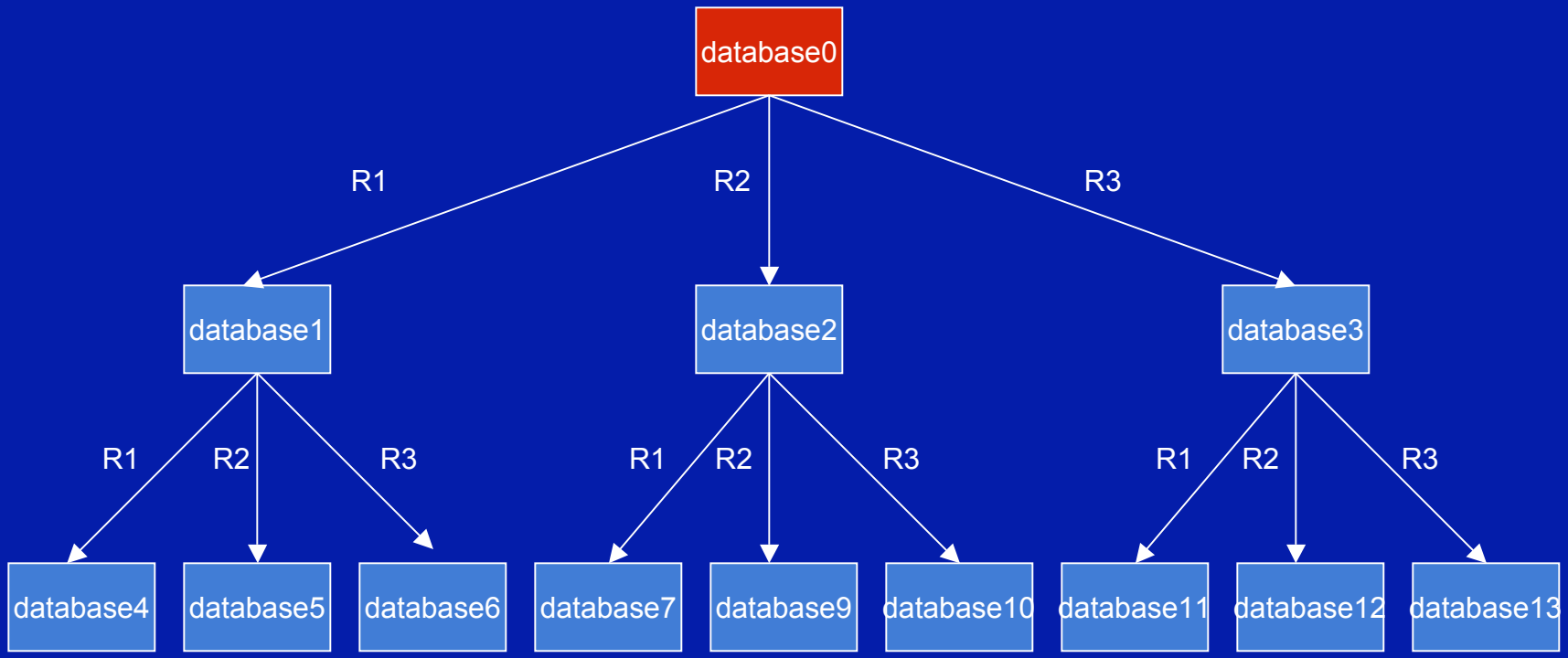
```
all_replacements(Board,Oldseq,Newseq,Y) :-
          bagof(X,replace_subseq(Board,Oldseq,Newseq,X),Y).

all_replacements(Board,Oldseq,Newseq,[]).

replace_subseq(Oldseq,Oldsubseq,Newsubseq,Newseq) :- append(L1,L2,Oldseq) ,
                                                     append(Oldsubseq,L3,L2) ,
                                                     append(Newsubseq,L3,L4) ,
                                                     append(L1,L4,Newseq).


append([],Z,Z).
append([A|X],Y,[A|Z]) :- append(X,Y,Z).
```

Rule-based systems and search

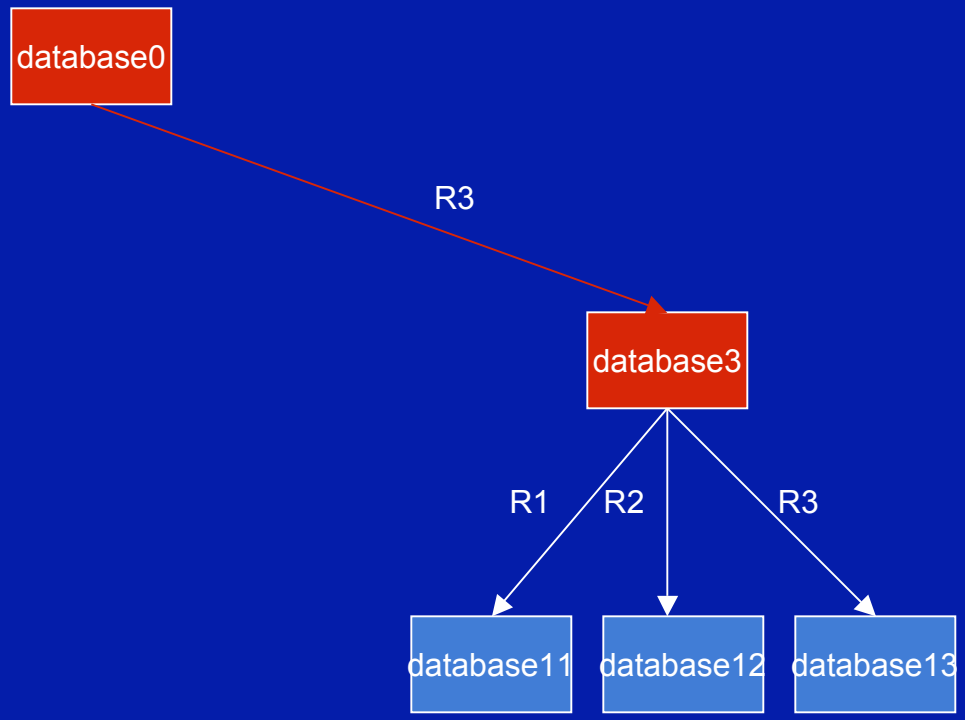
Rule-based systems and search



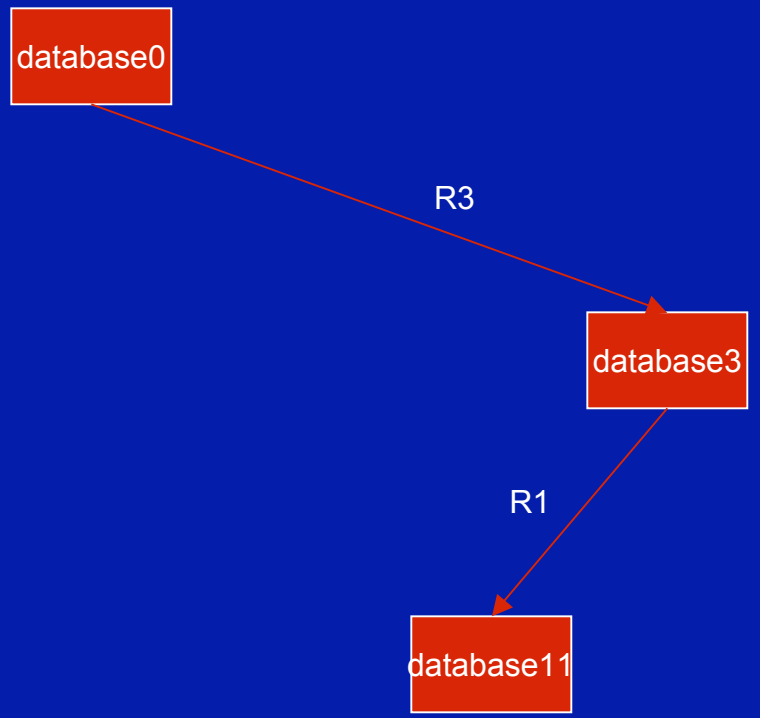
Rule-based systems and search



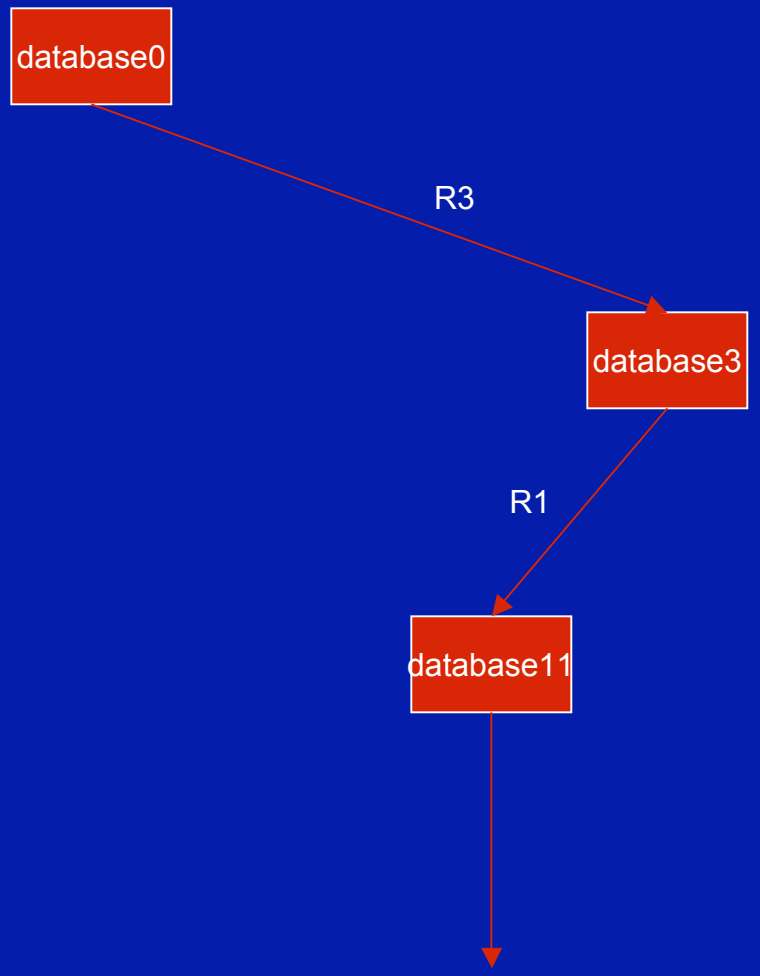
Rule-based systems and search



Rule-based systems and search



Rule-based systems and search



and so on...

Chapter 6

Chapter 6 in your textbook starts out by talking about expert systems, but it quickly turns into a discussion of how to build the CILOG meta-interpreter

Rule-based systems are similar in some ways to deductive reasoning systems, but rule-based systems aren't restricted to logical inference

Still, we could write simple rule-based systems in CILOG if CILOG could write to a data base, but no...

...so we'll wave goodbye to expert systems and move ahead to Chapter 8, which is about...

Actions and Planning

There are a number of situations where you'd like your man-made artifacts to have some ability to make a plan and then act according to the plan

Actions and Planning

Planners that can do stuff like this....

Actions and Planning

Actions and Planning

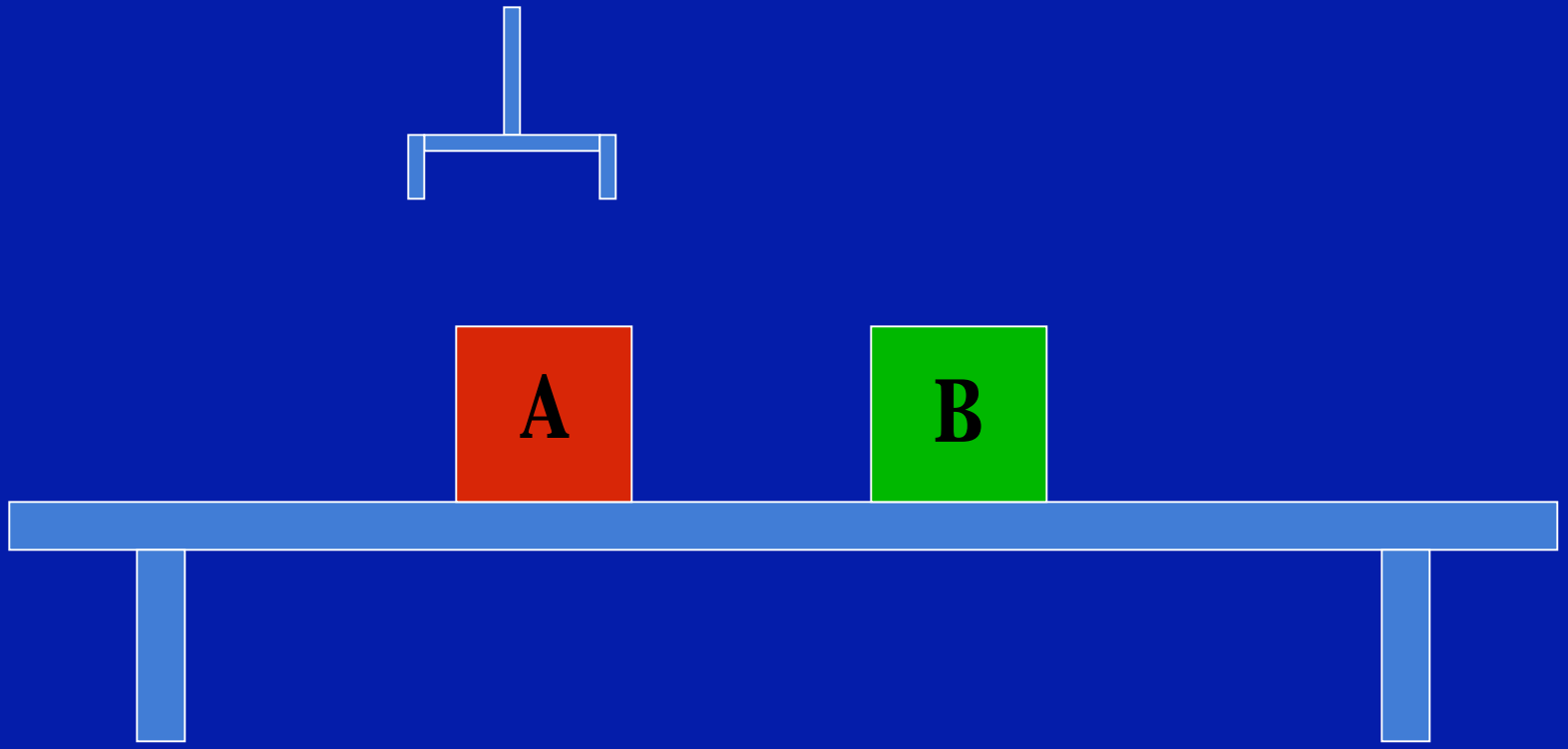
...have their beginnings in stuff like this...

Actions and Planning

...have their beginnings in stuff like this...

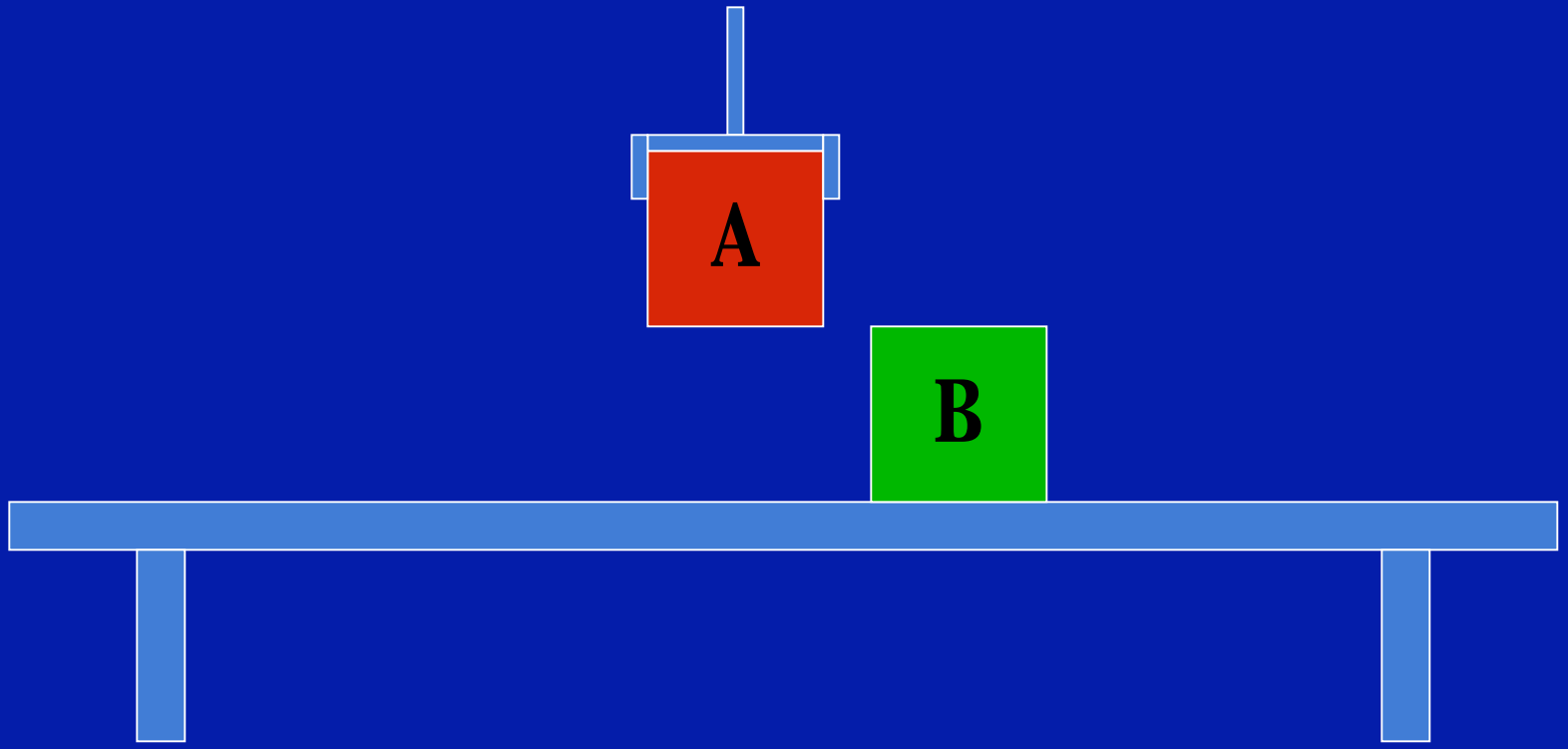
Actions and Planning

So we'll start at the beginning, and try to figure out how to get a computer to generate a plan to get from here....



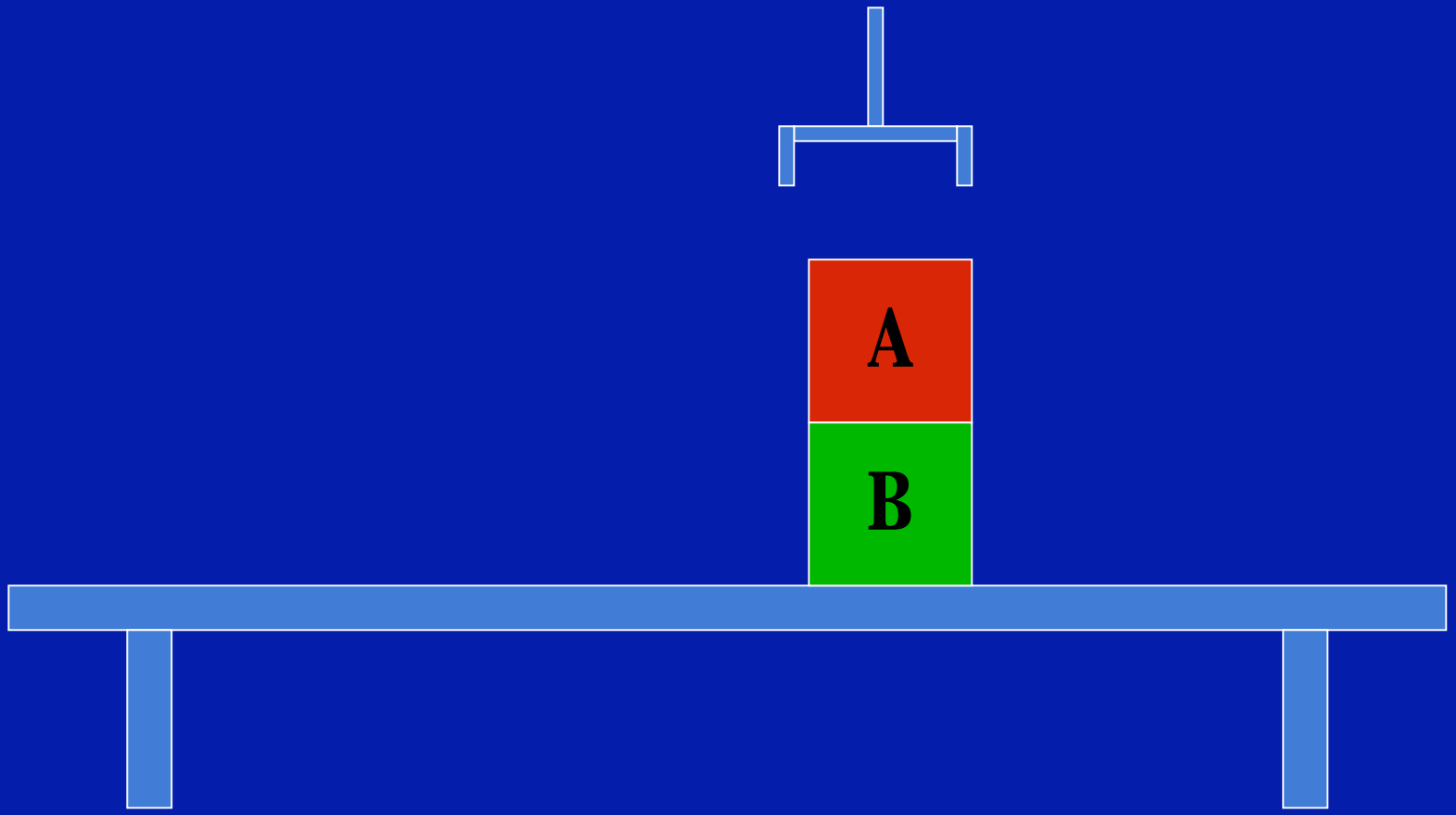
Actions and Planning

...to here...



Actions and Planning

...and finally to here



Actions and Planning

Start reading chapter 8