



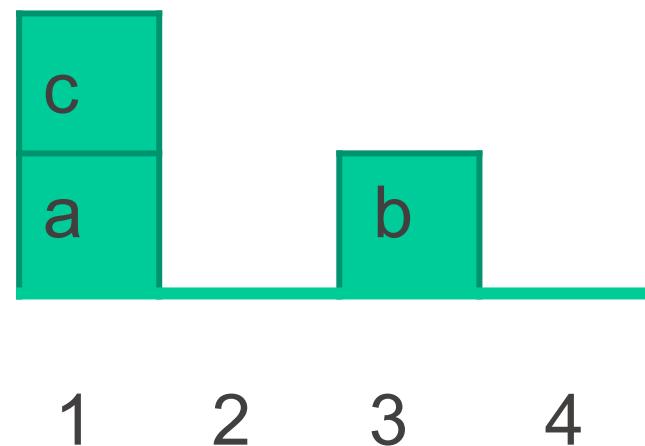
Planning

Lesson Overview (Agenda)

In the following lesson, we will explore:

1. Applications of Knowledge Representation
2. Planning
3. Breadth-first vs Depth-first planning
4. Prolog Programming

A BLOCKS WORLD PROBLEM



- Three blocks `block(a)`, `block(b)`, `block(c)`
- four locations `location(1)`, `location(2)`, `location(3)`, `location(4)`
- Relationships in initial state:
 - `on(block(c),block(a))`, `on(block(a),location(1))`, `on(block(b),location(3))`,
 - `clear(location(2))`, `clear(location(4))`, `clear(block(b))`, `clear(block(c))`
- Goal of plan e.g. build stack a, b, c : Goal stated as: `(on(block(a),block(b)), on(block(b),block(c)))`

Action schema

- Represents a number of actions by using variables
- move(X, Y, Z)

X stands for any block

Y, Z stand for any block or location

Situation Calculus Blocks World

Action: move(X,Y,Z)
Fluents: clear, on

Initial State:

block_exists(block(a)).
block_exists(block(b)).
block_exists(block(c)).

location_exists(location(1)).
location_exists(location(2)).
location_exists(location(3)).
location_exists(location(4)).

clear(location(2),[]).
clear(location(4),[]).
clear(block(b),[]).
clear(block(c),[]).
on(block(a), location(1),[]).
on(block(b), location(3),[]).
on(block(c), block(a),[]).

Sitcalc precondition axiom

%Let's add comments to this code together

```
poss([move(Block,From,To)|S]):-
    block_exists(Block),
    clear(Block,S),
    (location_exists(To) ; block_exists(To)),
    Block \= To,
    clear(To,S),
    (location_exists(From);block_exists(From)),
    on(Block,From,S).
```

SitCalc successor state axioms

% Let's add comments to this code together

clear(X,[move(Z,X,Y)|S]) :- poss([move(Z,X,Y)|S]).

clear(X,[A|S]) :-

 poss([A|S]),

 A != move(_,_,X),

 clear(X,S).

on(X,Y,[move(X,Z,Y)|S]) :- poss([move(X,Z,Y)|S]).

on(X,Y,[A|S]) :-

 poss([A|S]),

 A != move(X,Y,_),

 on(X,Y,S).

Sample Queries

```
?- on(block(b),location(2),S).  
S = [move(block(b), location(3), location(2))] .  
?-
```

Is planning that easy? Not so fast...

```
?- clear(location(3),S).  
ERROR: Stack limit (1.0Gb) exceeded
```

Depth-first search and Infinite Loops

```
[trace] ?- clear(location(3),S).
```

What's Happening?
Location(3) is clear if we move something from it

```
Call: (10) clear(location(3), _3838) ? creep
```

Can we move a block?

```
Call: (11) poss([move(_5048, location(3), _5052)|_5030]) ? creep
```

```
Call: (12) block_exists(5048) ? creep
```

Is block(a) a block?

```
Exit: (12) block_exists(block(a)) ? creep
```

Is block(a) clear?

```
Call: (12) clear(block(a), _5030) ? creep
```

Can we move a block from block(a)?

```
Call: (13) poss([move(_8080, block(a), _8084)|_8062]) ? creep
```

No progress since 12

```
Call: (14) block_exists(_8080) ? creep
```

Is block(a) a block?

```
Exit: (14) block_exists(block(a)) ? creep
```

Is block(a) clear? (in infinite loop)

```
Call: (14) clear(block(a), _8062) ? creep
```

```
Call: (15) poss([move(_11112, block(a), _11116)|_11094]) ? creep
```

```
Call: (16) block(_11112) ? creep
```

In this case, Prolog is descending an infinitely long branch of the search tree

```
Exit: (16) block_exists(block(a)) ? creep
```

```
Call: (16) clear(block(a), _11094) ? creep
```

```
Call: (17) poss([move(_14144, block(a), _14148)|_14126]) ? creep
```

```
Call: (18) block_exists(_14144) ? creep
```

```
Exit: (18) block_exists(block(a)) ? creep
```

```
Call: (18) clear(block(a), _14126) ? creep
```

```
Call: (19) poss([move(_17176, block(a), _17180)|_17158]) ?
```

Breadth First Planning

We are searching a tree of plans.

Depth first plans include an infinite sequence of moving a block back and forth.

We can fix the depth-first infinite loop problem by arranging for a breadth first search

We first generate all plans of length 1, until one of them satisfies the goal

Once plans of length 1 are exhausted, we move on to plans of length 2

And so on, until the goal succeeds, then we have our plan.

Breadth-first planning (cont'd)

First, try a plan of length 1

```
?- poss([A]),clear(location(3),[A]).  
A = move(block(b), location(3), location(2)) ;  
A = move(block(b), location(3), location(2)) ;  
A = move(block(b), location(3), location(2)) ;  
A = move(block(b), location(3), location(4)) ;  
A = move(block(b), location(3), location(4)) ;  
A = move(block(b), location(3), location(4)) ;  
A = move(block(b), location(3), block(c)) ;  
A = move(block(b), location(3), block(c)) ;  
A = move(block(b), location(3), block(c)) ;  
false.
```

Breadth-first planning (cont'd)

First, try a plan of length 2

?- poss([B,A]),clear(location(3),[B,A]).

B = move(block(b), location(2), location(4)),

A = move(block(b), location(3), location(2))

...etc...

We can implement breadth-first planning

```
% plan for a goal by doing a breadth-first search for plans  
plan(Goal,Plan):-bposs(Plan),Goal.
```

```
bposs(S) :- tryposs([],S).
```

```
tryposs(S,S) :- poss(S).           %S is a plan; otherwise..  
tryposs(X,S) :- tryposs([_|X],S). %increase plan length
```

Implement breadth-first planning (cont'd)

```
?- plan(clear(location(3),S),S).  
S = [move(block(b), location(3), location(2))]
```

Time to check your learning!

Let's see how many key concepts from Planning you recall by answering the following questions!

Why does a "normal" planning query not always terminate?

How does a breadth-first planning query address that problem?