POPL2 class (2020-05-06)

Searching in state space

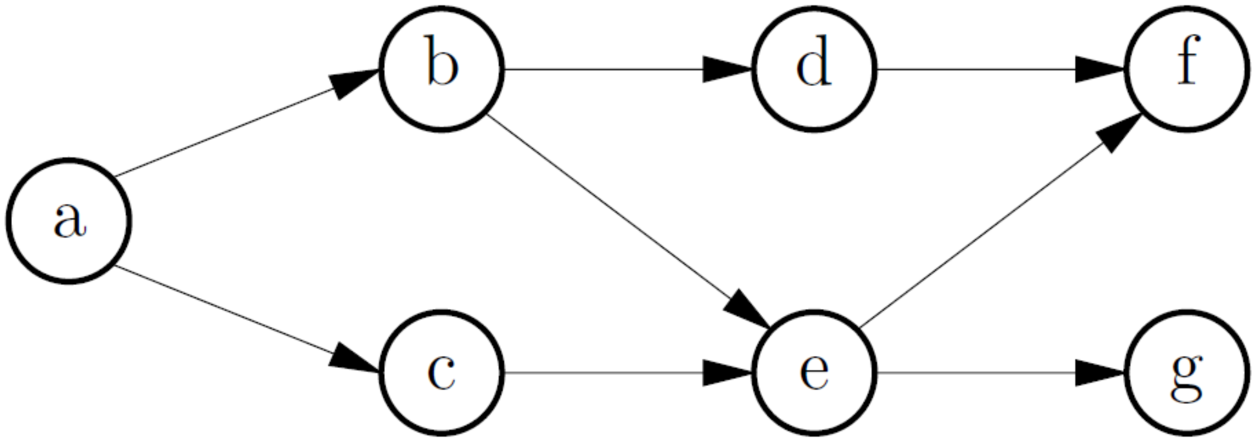
State-spaces and State-transitions

- Many problems in computer science can be formulated as a possibly innite set *S states* and a binary transition-relation
- Given some *start*-state s0 *in S* and a set *G sub S* of *goal*-states

$$s_0 \leadsto s_1,\ s_1 \leadsto s_2,\ s_2 \leadsto s_3,\ \cdots s_{n-1} \leadsto s_n \qquad\qquad s_n \in G$$

- *Planning* amounts to finding a sequence of worlds where the initial world is transformed into some desired final world.

State space



"To find a path from *X* to *Z*, first find an edge from *X* to *Y* and then nd a path from *Y* to *Z*".

$edge(a, b).$
$edge(a, c).$
$edge(b, d).$
$edge(b, e).$

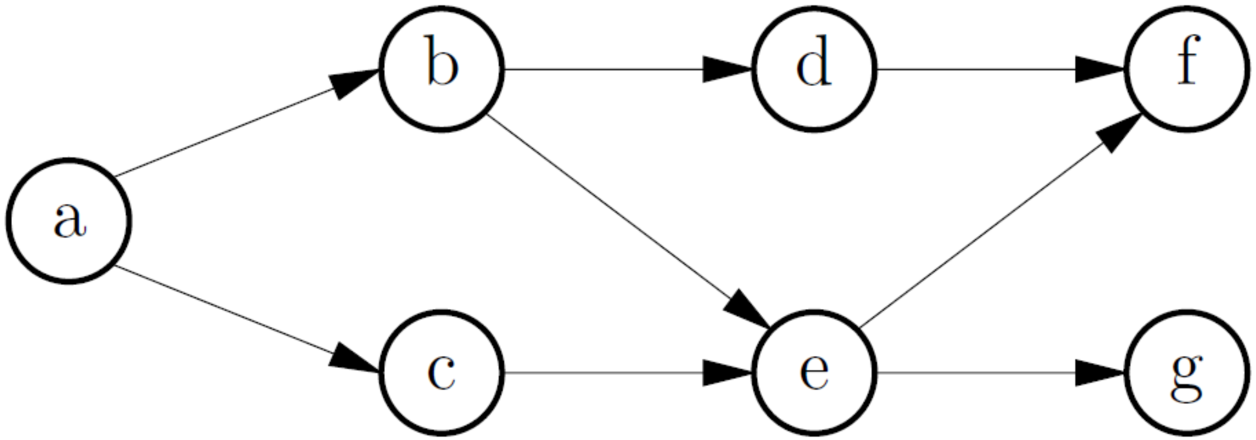
$edge(c, e).$
$edge(d, f).$
$edge(e, f).$
$edge(e, g).$

$$path(X, X).$$
$$path(X, Z) \leftarrow edge(X, Y), path(Y, Z).$$

$$path(X, Z) \leftarrow edge(Y, Z), path(X, Y).$$

Search in the backward direction

State space



$$goal\_state(f).$$
$$goal\_state(g).$$

$$path(X, X).$$
$$path(X, Z) \leftarrow edge(X, Y), path(Y, Z).$$

$$\leftarrow path(a, X), goal\_state(X).$$

$$path(\ulcorner goal \urcorner).$$
$$path(X) \leftarrow edge(X, Y), path(Y).$$

Loop detection

$$path(X, X).$$
$$path(X, Z) \leftarrow edge(X, Y), path(Y, Z).$$

$$edge(a, b). \quad edge(b, a). \quad edge(a, c).$$
$$edge(b, d). \quad edge(b, e). \quad edge(c, e).$$
$$edge(d, f). \quad edge(e, f). \quad edge(e, g).$$

Loop detection

$$path(X, X).$$
$$path(X, Z) \leftarrow edge(X, Y), path(Y, Z).$$

$$edge(a, b). \quad edge(b, a). \quad edge(a, c).$$
$$edge(b, d). \quad edge(b, e). \quad edge(c, e).$$
$$edge(d, f). \quad edge(e, f). \quad edge(e, g).$$

$$\begin{aligned}
&path(X, Y) \leftarrow \\
&\qquad path(X, Y, [X]).
\end{aligned}$$

$$\begin{aligned}
&path(X, X, Visited). \\
&path(X, Z, Visited) \leftarrow \\
&\qquad edge(X, Y), \\
&\qquad not\ member(Y, Visited), \\
&\qquad path(Y, Z, [Y|Visited]).
\end{aligned}$$

$$\begin{aligned}
&member(X, [X|Y]). \\
&member(X, [Y|Z]) \leftarrow \\
&\qquad member(X, Z).
\end{aligned}$$

Loop detection

$$path(X, X).$$
$$path(X, Z) \leftarrow edge(X, Y), path(Y, Z).$$

$$edge(a, b). \quad edge(b, a). \quad edge(a, c).$$
$$edge(b, d). \quad edge(b, e). \quad edge(c, e).$$
$$edge(d, f). \quad edge(e, f). \quad edge(e, g).$$

$$\begin{aligned} &path(X, Y, Path) \leftarrow \\ &\qquad path(X, Y, [X], Path). \end{aligned}$$

$$\begin{aligned} &path(X, X, Visited, Visited). \\ &path(X, Z, Visited, Path) \leftarrow \\ &\qquad edge(X, Y), \\ &\qquad not\ member(Y, Visited), \\ &\qquad path(Y, Z, [Y \mid Visited], Path). \end{aligned}$$

Water-jug problem

Two water jugs are given, a 4-gallon and a 3-gallon jug. Neither of them has any type of marking on it. There is an infinite supply of water (a tap?) nearby. How can you get exactly 2 gallons of water into the 4-gallon jug? Initially both jugs are empty.

Water-jug problem

Two water jugs are given, a 4-gallon and a 3-gallon jug. Neither of them has any type of marking on it. There is an infinite supply of water (a tap?) nearby. How can you get exactly 2 gallons of water into the 4-gallon jug? Initially both jugs are empty.

a state is described
by a pair *(x,y)* where *x* represents the amount of water in the 4-gallon jug
and *y* represents the amount of water in the 3-gallon jug. The start-state
then is *(0,0)*
and the goal-state is any pair where the first component equals 2

Water Jug problem

- empty the 4-gallon jug if it is not already empty;
- empty the 3-gallon jug if it is not already empty;
- fill up the 4-gallon jug if it is not already full;
- fill up the 3-gallon jug if it is not already full;
- if there is enough water in the 3-gallon jug, use it to fill up the 4-gallon jug until it is full;
- if there is enough water in the 4-gallon jug, use it to fill up the 3-gallon jug until it is full;
- if there is room in the 4-gallon jug, pour all water from the 3-gallon jug into it;
- if there is room in the 3-gallon jug, pour all water from the 4-gallon jug into it.

Water Jug problem

- empty the 4-gallon jug if it is not already empty;
- empty the 3-gallon jug if it is not already empty;
- fill up the 4-gallon jug if it is not already full;
- fill up the 3-gallon jug if it is not already full;

$$
\begin{aligned}
&action(X : Y, 0 : Y) \leftarrow X > 0.\\
&action(X : Y, X : 0) \leftarrow Y > 0.\\
&action(X : Y, 4 : Y) \leftarrow X < 4.\\
&action(X : Y, X : 3) \leftarrow Y < 3.
\end{aligned}
$$

Water Jug problem

- if there is enough water in the 3-gallon jug, use it to fill up the 4-gallon jug until it is full;
- if there is enough water in the 4-gallon jug, use it to fill up the 3-gallon jug until it is full;
- if there is room in the 4-gallon jug, pour all water from the 3-gallon jug into it;
- if there is room in the 3-gallon jug, pour all water from the 4-gallon jug into it.

$$
\begin{aligned}
&action(X:Y, 4:Z) \leftarrow X < 4, Z\ is\ Y - (4 - X), Z \geq 0.\\
&action(X:Y, Z:3) \leftarrow Y < 3, Z\ is\ X - (3 - Y), Z \geq 0.\\
&action(X:Y, Z:0) \leftarrow Y > 0, Z\ is\ X + Y, Z \leq 4.\\
&action(X:Y, 0:Z) \leftarrow X > 0, Z\ is\ X + Y, Z \leq 3.
\end{aligned}
$$

Water Jug problem

goal-state is known to be $\langle 2, X\rangle$ for any value of X

$$
\begin{aligned}
&path(X) \leftarrow \\
&\qquad path(0:0, [0:0], X).
\end{aligned}
$$

$$\text{goal} \leftarrow path(X)$$

$$X = [2:0, 0:2, 4:2, 3:3, 3:0, 0:3, 0:0].$$

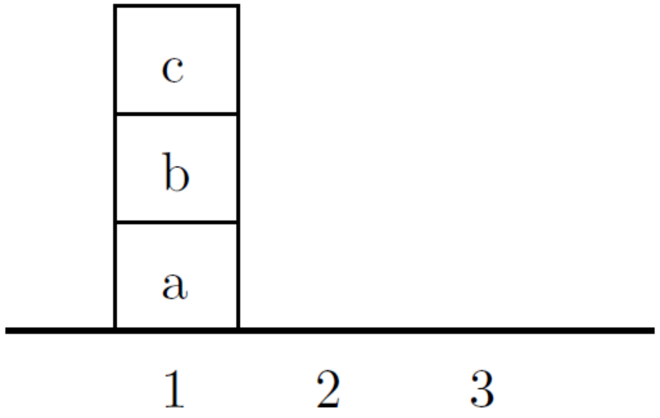
$$
\begin{aligned}
&path(2:X, Visited, Visited). \\
&path(State, Visited, Path) \leftarrow \\
&\qquad action(State, NewState), \\
&\qquad not\ member(NewState, Visited), \\
&\qquad path(NewState, [NewState|Visited], Path).
\end{aligned}
$$

$$
\begin{aligned}
&member(X, [X|Y]). \\
&member(X, [Y|Z]) \leftarrow \\
&\qquad member(X, Z).
\end{aligned}
$$

$$X = [2:0, 0:2, 4:2, 3:3, 3:0, 0:3, 4:3, 4:0, 0:0].$$

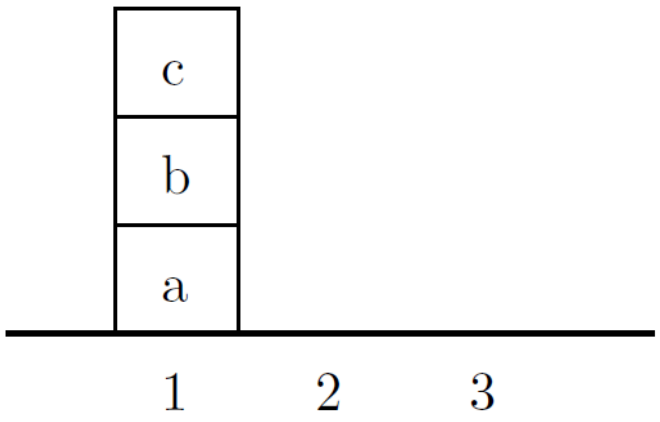
Block World

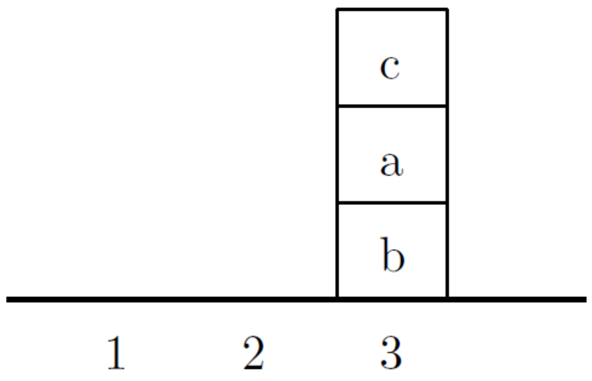
- table with three distinct positions
- blocks which may be stacked on top of each other
- move the blocks from a given start-state to a goal-state
- Only blocks which are free (that is, with no other block on top of them) can be moved.



Block World

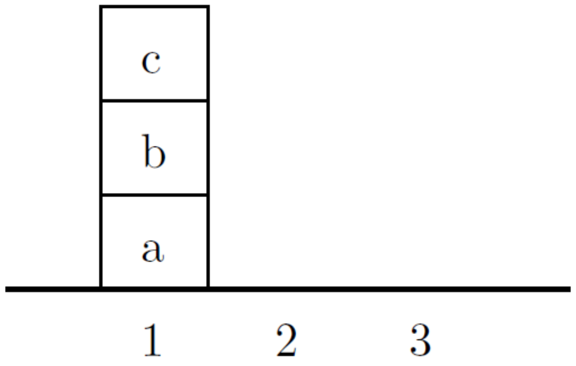
- *State/*3 to represent the three positions of the table
- *table* to denote the table.
- *on*(*X; Y*) the fact that *X* is positioned on top of *Y*
- *state*(*on*(*c; on*(*b; on*(*a; table*))); *table; table*)

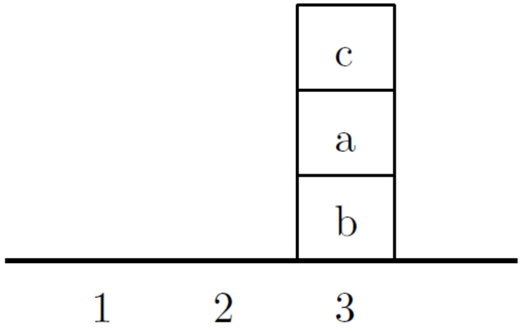




Block World

- if the first position is nonempty the topmost block can be moved to either the second or the third position;
- if the second position is nonempty the topmost block can be moved to either the first or the third position;
- if the third position is nonempty the topmost block can be moved to either the first or the second position.





Block World

- if the first position is nonempty the topmost block can be moved to either the second or the third position;
- if the second position is nonempty the topmost block can be moved to either the first or the third position;
- if the third position is nonempty the topmost block can be moved to either the first or the second position.

$$move(state(on(X, NewX), OldY, Z), state(NewX, on(X, OldY), Z)).$$
$$move(state(on(X, NewX), Y, OldZ), state(NewX, Y, on(X, OldZ))).$$

Block World

$$move(state(on(X, NewX), OldY, Z), state(NewX, on(X, OldY), Z)).$$
$$move(state(on(X, NewX), Y, OldZ), state(NewX, Y, on(X, OldZ))).$$

$$\begin{aligned} path(X, Y, Path) \leftarrow & \\ & path(X, Y, [X], Path). \end{aligned}$$

$$\begin{aligned} path(X, X, Visited, Visited)&. \\ path(X, Z, Visited, Path) \leftarrow & \\ & edge(X, Y), \\ & not\ member(Y, Visited), \\ & path(Y, Z, [Y|Visited], Path). \end{aligned}$$

$$\leftarrow path(\quad \begin{aligned} & state(on(c, on(b, on(a, table))), table, table), \\ & state(table, table, on(c, on(a, on(b, table)))), \quad X). \end{aligned}$$

$$X = [\quad \begin{aligned} & state(table, table, on(c, on(a, on(b, table)))), \\ & state(table, on(c, table), on(a, on(b, table))), \\ & state(on(a, table), on(c, table), on(b, table)), \\ & state(on(b, on(a, table)), on(c, table), table), \\ & state(on(c, on(b, on(a, table))), table, table) \quad] \end{aligned}$$