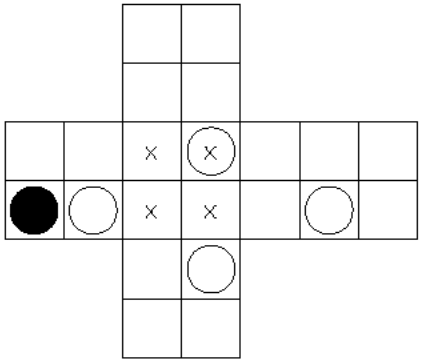
From Z Specification to Event-State based Reasoning (Introduction)

Jin Song Dong

Formally Specifying Shunting Games in Z Schemas

The figure below gives the board and starting position for a game of *Shunting*. A move consists of the black piece (the shunter) moving one position either vertically or horizontally provided either

- the position moved to is empty, or
- the position moved to is occupied by a white piece but the position beyond the white piece is empty, in which case the white piece is pushed into the empty position.



The shunter can not push two white pieces at the same time. At each stage a score is kept of the number of moves made so far. The game ends when the white pieces occupy the four positions marked with a cross.

Initial Specification Structure

Give a Z specification of this game. Hints:

$$Board == (1\,..\,7 \times 3\,..\,4) \cup (3\,..\,4 \times 1\,..\,6)$$
$$over == \{(3,3), (4,3), (3,4), (4,4)\}$$

$$next : Board \leftrightarrow Board$$
$$beyond : Board \times Board \nrightarrow \mathbb{N} \times \mathbb{N}$$
$$....$$

Shunting

$$bposn : Board$$
$$wposn : \mathbb{P}\, Board$$
$$score : \mathbb{N}$$
$$...$$

Shunting$_{\text{INIT}}$

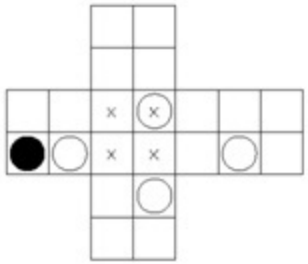
$$...$$

Move

$$...$$

Over

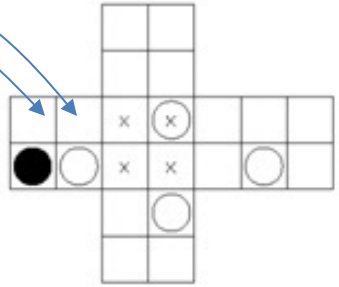
$$...$$



$$Board == (1 \mathinner{..} 7 \times 3 \mathinner{..} 4) \cup (3 \mathinner{..} 4 \times 1 \mathinner{..} 6)$$

$$next : Board \leftrightarrow Board$$

$$\begin{aligned}
&\forall (i, j), (k, l) : Board \bullet \\
&\quad (i, j) \ \underline{next} \ (k, l) \ \Leftrightarrow \\
&\qquad i = k \ \land \ (j = l + 1 \ \lor \ j = l - 1) \\
&\qquad \lor \\
&\qquad j = l \ \land \ (i = k + 1 \ \lor \ i = k - 1)
\end{aligned}$$

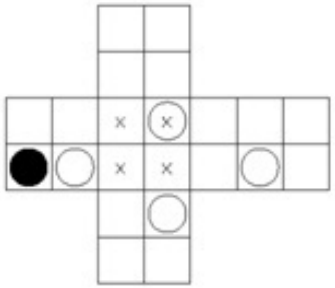


$$over : \mathbb{P}\, Board$$

$$over = \{(3, 3), (4, 3), (3, 4), (4, 4)\}$$

$$beyond : Board \times Board \pfun \mathbb{N} \times \mathbb{N}$$

$$\begin{aligned}
&\mathrm{dom}\, beyond = \{b, w : Board \mid b \ \underline{next} \ w\} \\
&\forall\, b, w : \mathrm{dom}\, beyond \bullet \\
&\qquad beyond(b, w) = 2w - b
\end{aligned}$$



Shunting

$bposn : Board$

$wposn : \mathbb{P}\, Board$

$score : \mathbb{N}$

$bposn \notin wposn$

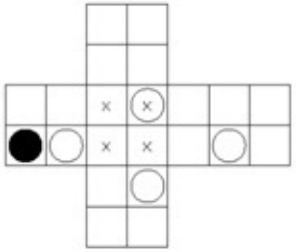
$\#wposn = 4$

Shunting$_{\text{INIT}}$

$bposn = (1, 3)$

$wposn = \{(2, 3), (4, 2), (4, 4), (6, 3)\}$

$score = 0$



Move

$\Delta Shunting$

$wposn \neq over$

$bposn'$ *next* $bposn$

$bposn' \notin wposn \Rightarrow wposn' = wposn$

$bposn' \in wposn \Rightarrow$

$\quad wposn' = (wposn - \{bposn'\}) \cup \{beyond(bposn, bposn')\}$

$score' = score + 1$

Over

$\Xi Shunting$

$score! : \mathbb{N}$

$wposn = over$

$score! = score$

- Z is good for capturing high level requirements, e.g., the rules of the game.
- Z specification focus on what the problem is, but not how to solve the problem.

- What about traces of move events?
- Is the goal state "over" reachable?
- If it is reachable, any bad moves will make it never reachable?
- What is the minimum moves to reach the goal state?

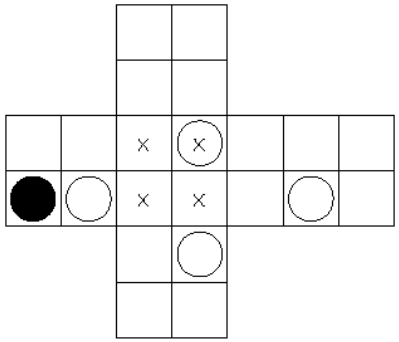
- Need Reasoning & Verification!

Shunting Example Revisited

- Can we develop a programming like model that can solve the puzzle automatically?

The figure below gives the board and starting position for a game of *Shunting*. A move consists of the black piece (the shunter) moving one position either vertically or horizontally provided either

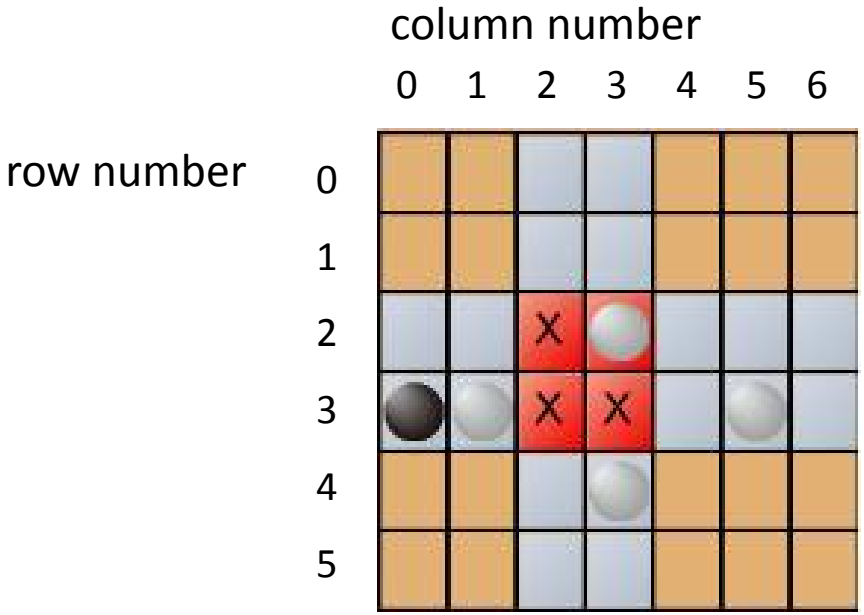
- the position moved to is empty, or
- the position moved to is occupied by a white piece but the position beyond the white piece is empty, in which case the white piece is pushed into the empty position.



In fact, let's make it more interesting that the black(dog) has limited energy and PushUp (the hill) requires extra energy.

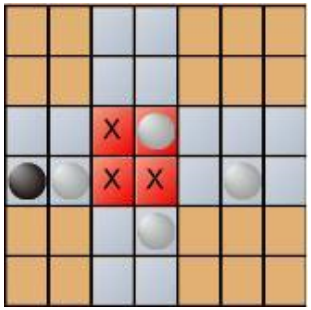
The shunter can not push two white pieces at the same time. At each stage a score is kept of the number of moves made so far. The game ends when the white pieces occupy the four positions marked with a cross.

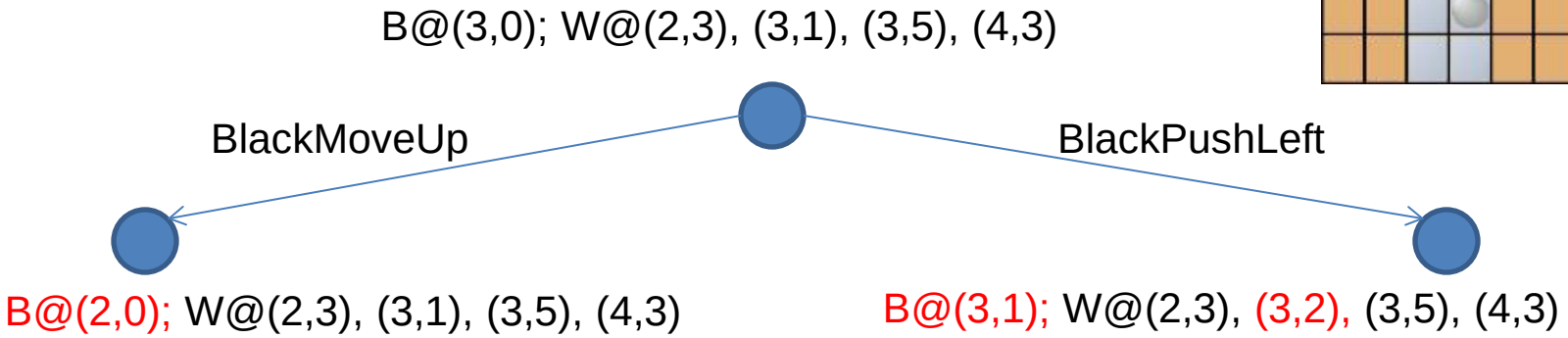
Example: Shunting Game



– A state consists of the positions of the black one and the white ones. Initially, it is
  - Black at (3,0); Whites at (2,3), (3,1), (3,5), (4,3)

Example: Shunting Game



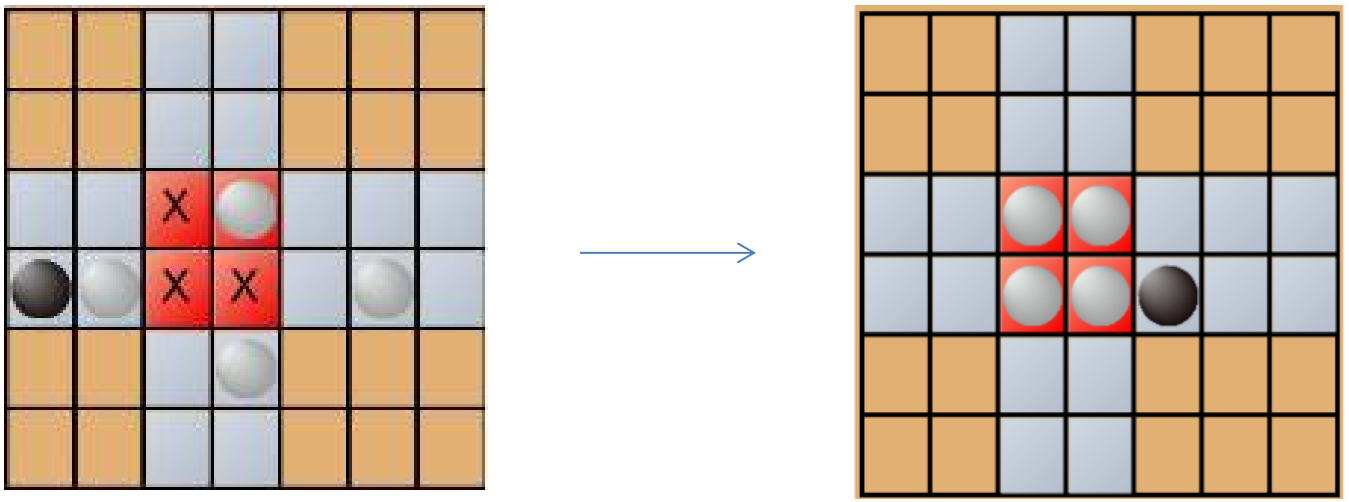


– A transition is caused by the movement of the black.

Reachability Analysis

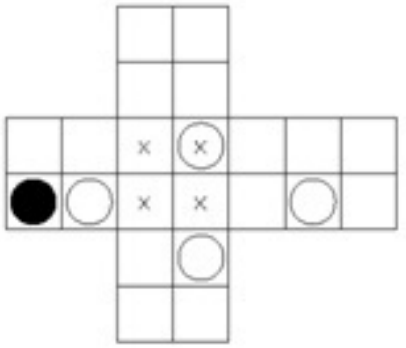
- Goal: to determine whether there is a reachable state such that certain condition is satisfied.
  - e.g., searching for a state such that

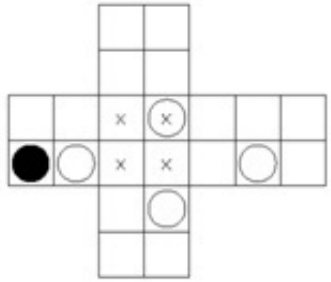
    the white ones are at (2,2), (2,3),(3,2), (3,3)



```
//The following are constants of the shunting game
#define M 7;
#define N 6;
#define o -1; //off board
#define a 1;  // available
#define w 0;  // white occupied
//       col number:  0 1 2 3 4 5 6
var board[N][M] = [o,o,a,a,o,o,o, //0 row number
                   o,o,a,a,o,o,o, //1
                   a,a,a,w,a,a,a, //2
                   a,w,a,a,a,w,a, //3
                   o,o,a,w,o,o,o, //4
                   o,o,a,a,o,o,o]; //5
// Black position:
var r = 3; var c = 0; work = 0;

Game = [work <= 10]okay -> ([r-1>=0]MoveUp [] [r-2>=0]PushUp
                            [] [r+1<N]MoveDown [] [r+2<N]PushDown
                            [] [c-1>=0]MoveLeft [] [c-2>=0]PushLeft
                            [] [c+1<M]MoveRight [] [c+2<M]PushRight)
        [] [work > 10]overworked -> Skip;
```



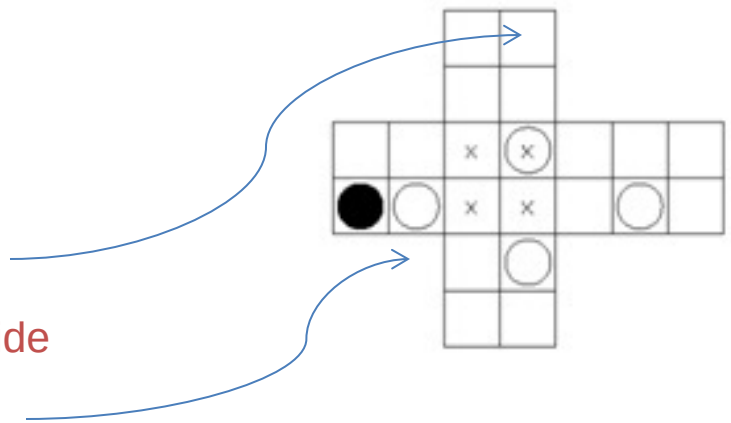


```
MoveUp = [board[r-1][c]==a]go_up{r=r-1} -> Game;
PushUp = [board[r-2][c]==a && board[r-1][c]==w]
      push_up{board[r-2][c]=w; board[r-1][c]=a; r=r-1;work=work+2} -> Game;

MoveDown = [board[r+1][c]==a]go_down{r=r+1} -> Game;
PushDown = [board[r+2][c]==a && board[r+1][c]==w]
            push_down{board[r+2][c]=w; board[r+1][c]=a; r=r+1} -> Game;

MoveLeft = [board[r][c-1]==a]go_left{c=c-1} -> Game;
PushLeft = [board[r][c-2]==a && board[r][c-1]==w]
            push_left{board[r][c-2]=w; board[r][c-1]=a; c=c-1} -> Game;

MoveRight = [board[r][c+1]==a]go_right{c=c+1} -> Game;
PushRight = [board[r][c+2]==a && board[r][c+1]==w]
            push_right{board[r][c+2]=w; board[r][c+1]=a; c=c+1} -> Game;
```



```
//one particular potential trouble position
#define trouble board[0][3] == w;

//testing if a white can be pushed to ourside
#define outside board[4][1] == w;

#assert Game reaches trouble;
#assert Game reaches outside;

#define goal board[2][2] == w && board[2][3] == w
             && board[3][2] == w  && board[3][3] == w;
#assert Game reaches goal;
#assert Game reaches goal with min(work);  //optimisation,
                                            //towards a problem solving tool

#assert Game |= [] (trouble -> !<> goal);
//show the trouble position will prevent the goal
```