Prolog: Containerschepen

Context

Proficiat! Je bent net aangenomen als Prolog programmeur bij een bedrijf dat containers in- en uitlaadt voor grote containerschepen. Je nieuwe baas vraagt je om een Prolog programma te schrijven dat het bedrijf helpt om een bepaalde container van een schip uit te laden.

Representatie

De siutatie op een containerschip kan visueel worden voorgesteld als een puzzel op een $W \times H$ grid:

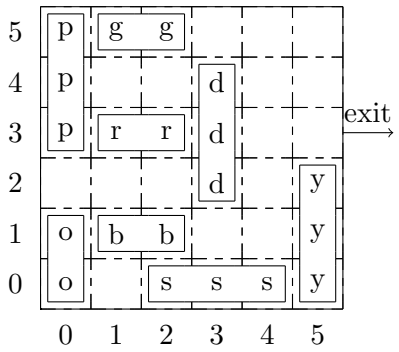


Fig. 1. Voorbeeld van een puzzel.

Hier duiden de blokjes met letters de containers aan. De lege blokjes zijn vrij. In Prolog kunnen we zo een puzzel voorstellen met een Prolog term `puzzle(Width, Height, Boxes)`:

```
Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]).
```

Hierbij duiden `Width` en `Height` de breedte en de hoogte aan van de grid. `Boxes` is in deze representatie een **geordende** lijst van `box(ID, Pos1, Pos2)` Prolog termen: `ID` is een **unieke** identifier (de kleur) en `Pos1` en `Pos2` duiden de eindpunten van de containers aan. **Belangrijk:** de coördinaat `Pos1` is steeds de meest linkse en meest onderste coördinaat. Je mag er dus van uit gaan dat `Pos1` en `Pos2` geordend zijn.

Opdracht 1.1: Schrijf een predicaat `box_contains(Box, Pos)`, dat gegeven een container `Box` en gegeven een positie `Pos`, slaagt indien de positie in de container ligt.

```
?- box_contains(box(green, (1, 5), (2, 5)), (1,5)).
true.
?- box_contains(box(green, (1, 5), (2, 5)), (3,3)).
false.
?- box_contains(box(purple, (0, 3), (0, 5)), (0,4)).
true.
?- box_contains(box(purple, (0, 3), (0, 5)), (1,4)).
false.
```

Opdracht 1.2: Schrijf een predicaat `box_at(Puzzle, Pos)`, dat gegeven een puzzel `Puzzle` en gegeven een positie `Pos`, slaagt indien er in de puzzel op positie `Pos` een container ligt.

```
% All examples use the puzzle of figure 1.
?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   box_at(Puzzle, (3,2)).
true.
?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   box_at(Puzzle, (2,2)).
false.
?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   box_at(Puzzle, (5,5)).
false.
```

Containers verplaatsen

Containers kunnen verplaatst worden. Een horizontale container kan 1 vakje naar rechts (`right`) of links (`left`) bewegen. Verticale containers kunnen dit naar boven (`up`) of beneden (`down`). Horizontale containers kunnen dus niet naar boven of onder bewegen en verticale containers niet naar links of rechts. Een container kan ook niet door een andere container bewegen of van het spelbord bewegen.

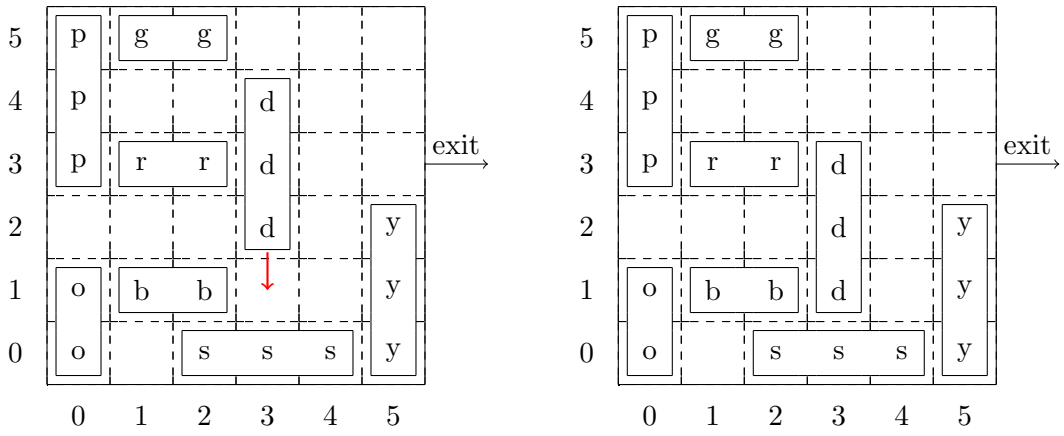


Fig. 2. Voorbeeld van het verplaatsen van container `dark (d)` naar beneden (`down`).

Opdracht 2: Schrijf een predicaat `move_box(Puzzle, Box, Dir, NewPuzzle)` dat gegeven een container `Box` in een puzzel `Puzzle`, `Dir` unificeert met een mogelijke geldige verplaatsing van de container. `NewPuzzle` unificeert met de puzzle waarbij de container **exact één** vakje verplaatst is. Vergeet niet de lijst van nieuwe containers opnieuw te sorteren. Indien er geen geldige verplaatsingen zijn voor de container, faalt het predicaat. **Opmerking:** De volgorde van de richtingen bij backtracken is niet van belang. Hiermee wordt bij de testen rekening gehouden.

```
?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   move_box(Puzzle, box(dark, (3,2), (3,4)), Dir, NewPuzzle).
Dir = up,
NewPuzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,3), (3,5)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]);
Dir = down,
NewPuzzle = puzzle(6, 6,
```

```
     [box(blue, (1,1), (2,1)), box(dark, (3,1), (3,3)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]);
false.  % dark is a vertical container, so only up/down.

?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   move_box(Puzzle, box(green, (1,5), (2,5)), Dir, _).
Dir = right;
false.  % green cannot move left because of purple

?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   move_box(Puzzle, box(orange, (0,0), (0,1)), Dir, _).
Dir = up;
false.  % orange cannot move down, because down is off the board.
```

Stopconditie

De puzzel is opgelost eens de container met de letter **r** (de rode container) vrij naar rechts kan bewegen richting de uitgang. De uitgang is steeds helemaal rechts op dezelfde rij als de rode container. Je mag er dus van uitgaan dat er altijd een rode container is en dat die bovendien altijd horizontaal is. Een voorbeeld van een opgeloste puzzel is hieronder te zien:

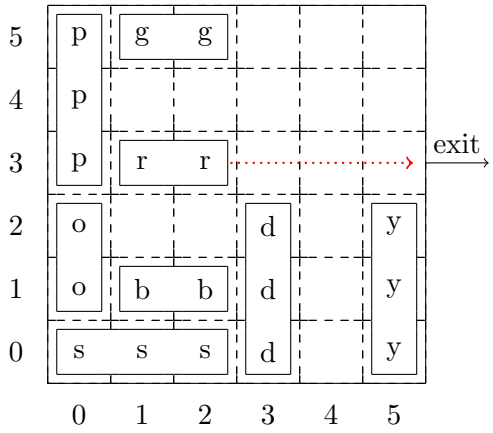


Fig. 3. Voorbeeld van de puzzel in figuur 1 opgelost.

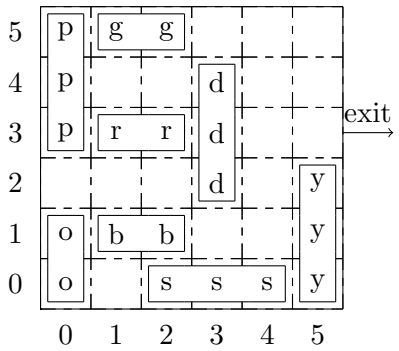
Opdracht 3: Schrijf het predicaat `finished(Puzzle)` dat slaagt indien de gegeven puzzel opgelost is. **Hint:** maak gebruik van het predicaat `move_box/4` van de vorige opgave.

```
?- Puzzle = puzzle(6, 6,[ box(red, (1,3), (2,3))]),
   finished(Puzzle).
true. % nothing in the way

?- Puzzle = puzzle(6, 6,
      [ box(red, (1,3), (2,3)) , box(yellow, (5,1), (5,3))]),
   finished(Puzzle).
false. % yellow in the way

% Puzzle from figure 3.
?- Puzzle = puzzle(6, 6,
     [box(blue, (1,1), (2,1)), box(dark, (3,0), (3,2)),
      box(green, (1,5), (2,5)), box(orange, (0,1), (0,2)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (0,0), (2,0)), box(yellow, (5,0), (5,2))]),
   finished(Puzzle).
true. % nothing in the way

% Puzzle from figure 1.
?- Puzzle = puzzle(6, 6,
```

```
     [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
      box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
      box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
      box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   finished(Puzzle).
false. % no, because dark is in the way.
```

Solver

Het doel van de puzzel is dus om de weg voor de **r** container vrij te maken richting de uitgang. We proberen dit te doen met maximaal N verplaatsingen (zie Opdracht 2). Neem bijvoorbeeld de puzzel van figuur 1:



Deze puzzel kan opgelost worden met exact 5 verplaatsingen/zetten. Één mogelijke oplossing is bijvoorbeeld:

Zet	ID	Richting	`ID-Direction` term
1	`orange`	`up`	`orange-up`
2	`silver`	`left`	`silver-left`
3	`silver`	`left`	`silver-left`
4	`dark`	`down`	`dark-down`
5	`dark`	`down`	`dark-down`

Opdracht 4: Schrijf het predicaat `solve(Puzzle, MaxDepth, Solution)` dat een puzzel oplost in maximaal `MaxDepth` stappen waarbij `MaxDepth` gegeven is. Hierbij moet `Solution` unificeren met een lijst van `ID-Direction` Prolog termen zoals in de bovenstaande tabel. **Opmerking:** Indien er verschillende oplossingen zijn, volstaat het om slechts één oplossing te geven. De testen houden hier rekening mee.

```
?- Puzzle = puzzle(6, 6, [box(red, (1,3), (2,3)) , box(yellow, (5,1), (5,3))]),
   solve(Puzzle, 1, Solution).
Solution = [yellow-down];
```

```
false.

?- Puzzle = puzzle(6, 6,
       [box(blue, (4, 3), (4, 4)), box(red, (1,3), (2,3)),
       box(yellow, (5,1), (5,3))]),
   solve(Puzzle, 1, Solution).
% This puzzle needs at least 2 moves to solve.
false.

?- Puzzle = puzzle(6, 6,
       [box(blue, (4, 3), (4, 4)), box(red, (1,3), (2,3)),
       box(yellow, (5,1), (5,3))]),
   solve(Puzzle, 2, Solution).
% it suffices to only return one of these:
Solution = [blue-up, yellow-down];
Solution = [yellow-down, blue-up];
false.

% Puzzle from figure 1.
?- Puzzle = puzzle(6, 6,
        [box(blue, (1,1), (2,1)), box(dark, (3,2), (3,4)),
         box(green, (1,5), (2,5)), box(orange, (0,0), (0,1)),
         box(purple, (0,3), (0,5)), box(red, (1,3), (2,3)),
         box(silver, (2,0), (4,0)), box(yellow, (5,0), (5,2))]),
   solve(Puzzle, 5, Sol).

% it suffices to only return one of these:
Sol = [dark-down,orange-up,silver-left,silver-left,dark-down];
Sol = [dark-down,silver-left,orange-up,silver-left,dark-down];
Sol = [orange-up,dark-down,silver-left,silver-left,dark-down];
Sol = [orange-up,silver-left,dark-down,silver-left,dark-down];
Sol = [orange-up,silver-left,silver-left,dark-down,dark-down];
Sol = [silver-left,dark-down,orange-up,silver-left,dark-down];
Sol = [silver-left,orange-up,dark-down,silver-left,dark-down];
Sol = [silver-left,orange-up,silver-left,dark-down,dark-down];
false.
```

Veel succes!