A Tabled Prolog Program for Solving Sokoban

Neng-Fa Zhou Agostino Dovier

Department of Computer and Information Science,
CUNY Brooklyn College & Graduate Center, USA

Department of Mathematics and Computer Science,
University of Udine, Italy

Pescara, September 2nd, 2011

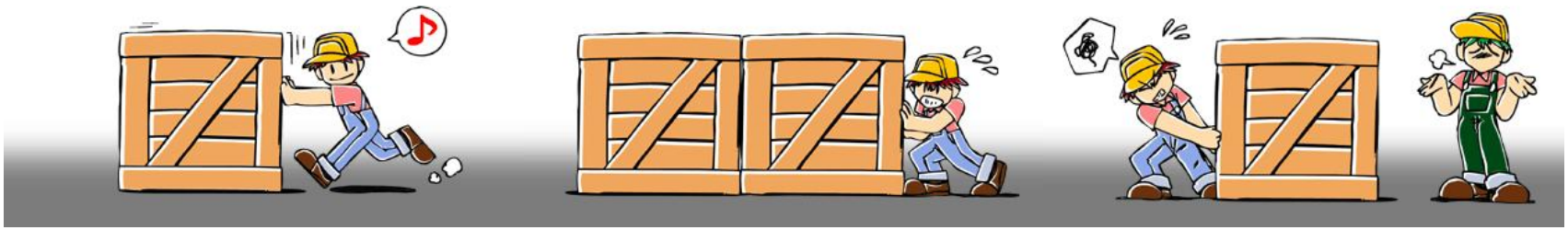
Summary

History

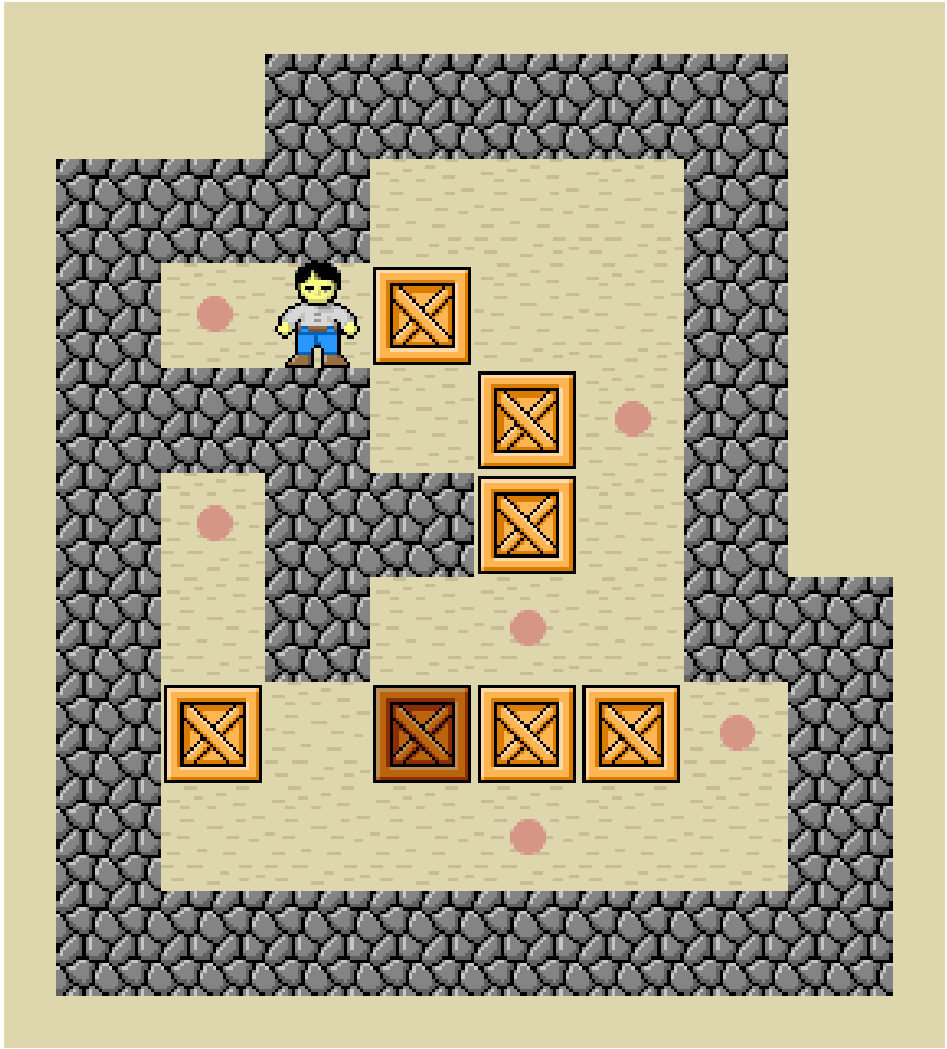
- Sokoban is a type of transport puzzle invented by *Hiroyuki Imabayashi* in 1980
- Published by the Japanese company Thinking Rabbit, Inc. in 1982.
- Sokoban means “warehouse-keeper” (magazziniere) in Japanese.
- Thinking Rabbit joined Square Co., Ltd.

Sokoban Rules

(from `http://www.sokoban.jp/`)

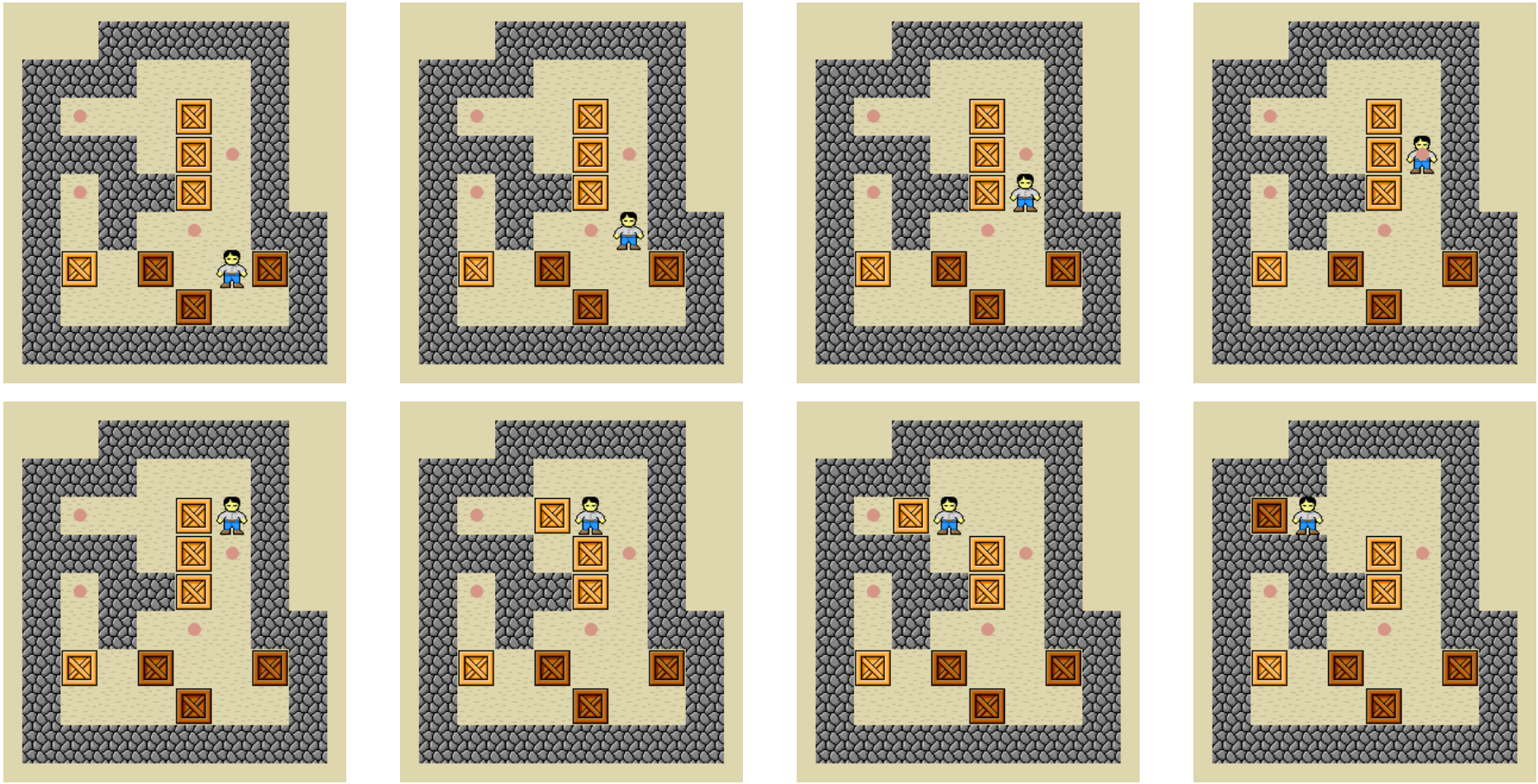


Sokoban @ work



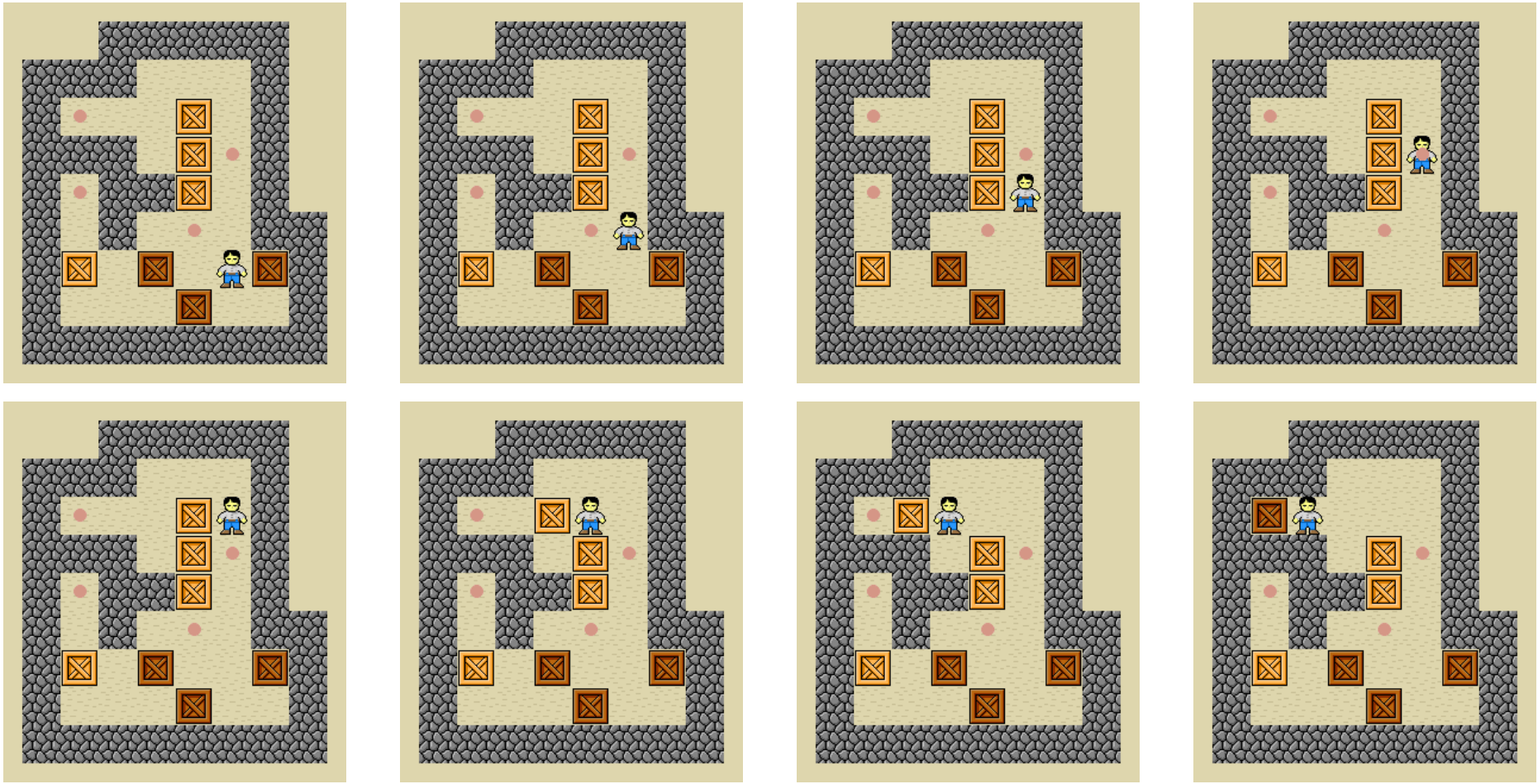
Macro-actions

The following sequence of 8 actions



Macro-actions

The following sequence of 8 actions



is counted as a single (macro) action.

The genesis of this work

- With Andrea Formisano and Enrico Pontelli we developed a CLP(FD) solver for the action description language $\mathcal{B}$ (and a compiler in ASP) [ICLP07–MG65]

The genesis of this work

- With Andrea Formisano and Enrico Pontelli we developed a CLP(FD) solver for the action description language $\mathcal{B}$ (and a compiler in ASP) [ICLP07–MG65]
- Neng-Fa Zhou is the main developer (actually, the father) of B-Prolog that includes a fast solver of constraints on finite domains

The genesis of this work

- With Andrea Formisano and Enrico Pontelli we developed a CLP(FD) solver for the action description language $\mathcal{B}$ (and a compiler in ASP) [ICLP07–MG65]
- Neng-Fa Zhou is the main developer (actually, the father) of B-Prolog that includes a fast solver of constraints on finite domains
- In the 2009 ASP competition we wrote with him some $\mathcal{B}$ domains that, once interpreted with the B-Prolog solver behaved very well (in particular peg-solitaire).

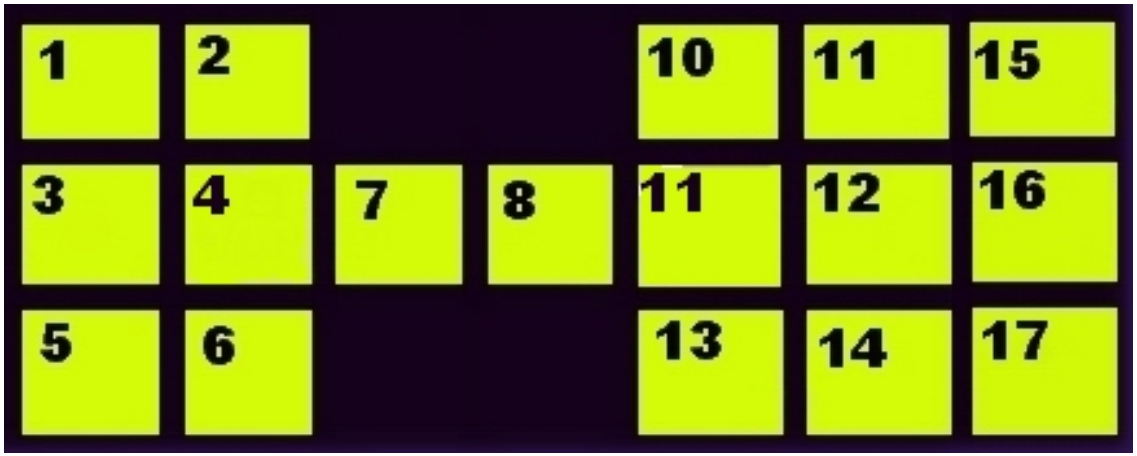
The genesis of this work

- With Andrea Formisano and Enrico Pontelli we developed a CLP(FD) solver for the action description language $\mathcal{B}$ (and a compiler in ASP) [ICLP07–MG65]
- Neng-Fa Zhou is the main developer (actually, the father) of B-Prolog that includes a fast solver of constraints on finite domains
- In the 2009 ASP competition we wrote with him some $\mathcal{B}$ domains that, once interpreted with the B-Prolog solver behaved very well (in particular peg-solitaire).
- Neng-Fa asked us to do the same for the 2011 competition.

The genesis of this work

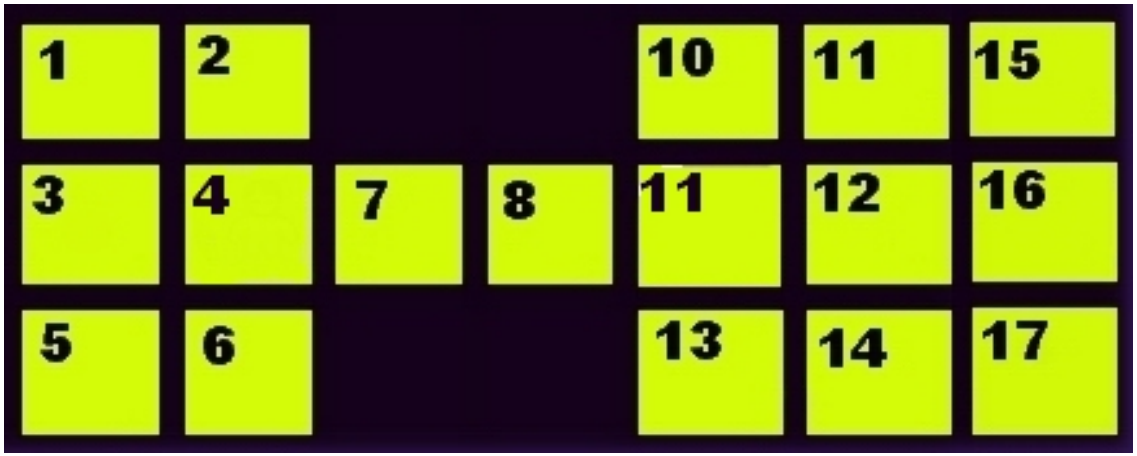
- With Andrea Formisano and Enrico Pontelli we developed a CLP(FD) solver for the action description language $\mathcal{B}$ (and a compiler in ASP) [ICLP07–MG65]
- Neng-Fa Zhou is the main developer (actually, the father) of B-Prolog that includes a fast solver of constraints on finite domains
- In the 2009 ASP competition we wrote with him some $\mathcal{B}$ domains that, once interpreted with the B-Prolog solver behaved very well (in particular peg-solitaire).
- Neng-Fa asked us to do the same for the 2011 competition.
- This approach for Sokoban was un-successful, but this forced us to look for another declarative approach.

Representation




```
location(1). ... location(17).
```

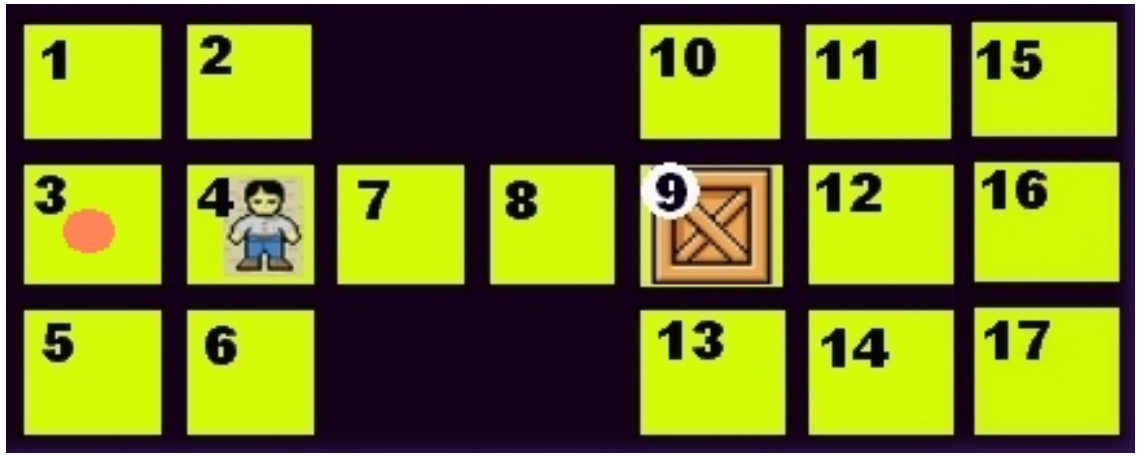
Representation




```
location(1). ... location(17).

step(1,right,2). step(10,right,11). step(11,right,15).
step(2,left,1).  step(11,left,10).  step(15,left,11).
step(1,down,3).  step(2,down,4).
step(3,up,1).    step(4,up,2).
...
```

Representation



Encoding in B

Fluents

```
fluent(free(L)) :-
   location(L).
fluent(box_in(L)) :-
   location(L).
fluent(sokoban_in(L)) :-
   location(L).
```

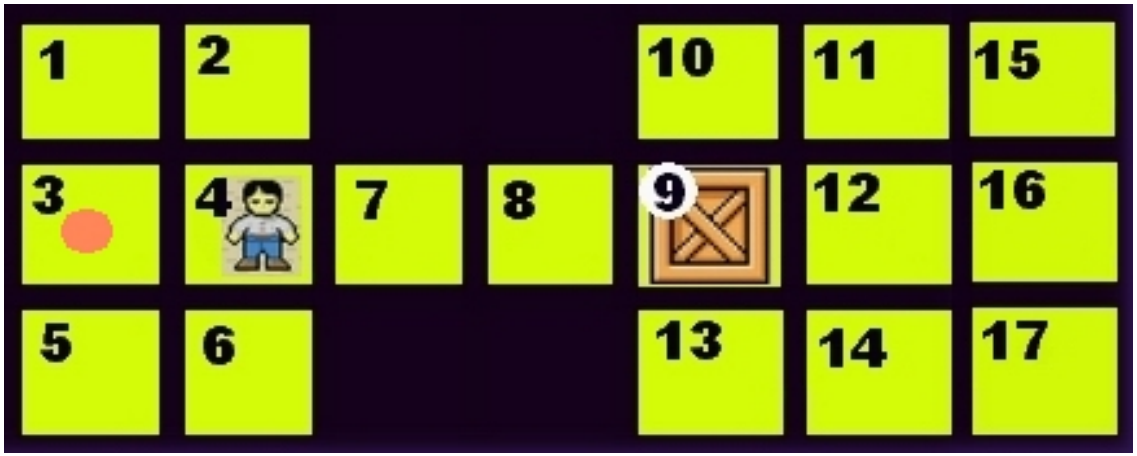
Encoding in B

Fluents

```
fluent(free(L)) :-
   location(L).
fluent(box_in(L)) :-
   location(L).
fluent(sokoban_in(L)) :-
   location(L).

fluent(reachable(A)) :-
    location(A).
```

Representation

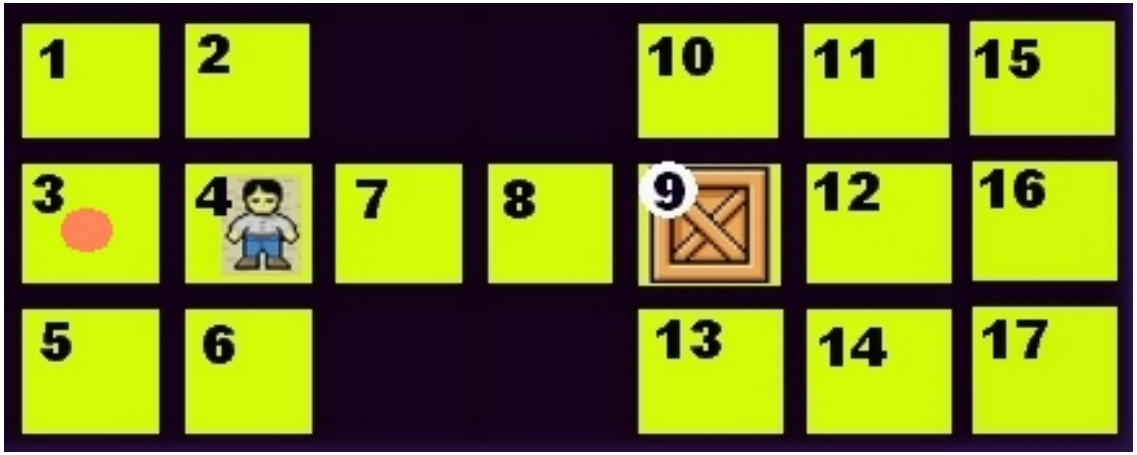



```
free(1). free(2). free(10). free(11). free(15).
free(7). free(8). free(12). free(16).
free(5). free(6). free(13). free(14). free(17).

box_in(9).

sokoban_in(4).
```

Representation




```
free(1). free(2). free(10). free(11). free(15).
free(7). free(8). free(12). free(16).
free(5). free(6). free(13). free(14). free(17).

box_in(9).

sokoban_in(4).

goal(box_in(3)).
```

Input from ASP competition

```
top(col4row4,col4row3). top(col4row3,col4row2).
right(col4row2,col5row2). right(col5row2,col6row2).
....
```

Input from ASP competition

```
top(col4row4,col4row3). top(col4row3,col4row2).
right(col4row2,col5row2). right(col5row2,col6row2).
....

box(col8row2). box(col3row4).
sokoban(col4row4).
```

Input from ASP competition

```
top(col4row4,col4row3). top(col4row3,col4row2).
right(col4row2,col5row2). right(col5row2,col6row2).
....

box(col8row2). box(col3row4).
sokoban(col4row4).


solution(col3row2). solution(col2row2).
```

Encoding in B

Actions (help from Andrea Formisano)

```
action(push(From,D,To)) :-
    location(From), location(To), neq(From,To),
    direction(D), % D = left, right, up, down
    step(_Sokoban,D,From),
    straight_connection(From,To,D,_).
```

Encoding in B

Actions (help from Andrea Formisano)

```
action(push(From,D,To)) :-
    location(From), location(To), neq(From,To),
    direction(D), % D = left, right, up, down
    step(_Sokoban,D,From),
    straight_connection(From,To,D,_).

executable(push(From,D,To),[sokoban_in(S0),reachable(S1),
                            box_in(From) | Free_LIST ]) :-
    action(push(From,D,To)),
    location(S0),location(S1),
    step(S1,D,From),
    straight_connection(From,To,D,[From|PATH]),
    empty_path(PATH, Free_LIST ).

empty_path([],[]).
empty_path([L|R],[free(L)|S]) :-
      empty_path(R,S).
```

Encoding in B

Actions Effects

```
causes(push(From,D,To), box_in(To) ,[]) :-
   action(push(From,D,To)).

causes(push(From,D,To), neg(box_in(From)) ,[]) :-
   action(push(From,D,To)).

causes(push(From,D,To),sokoban_in(S),[]) :-
   action(push(From,D,To)),
   location(S), step(S,D,To).

causes(push(From,D,To),free(S),[sokoban_in(S)]) :-
   action(push(From,D,To)),
   location(S), \+ step(S,D,To).

causes(push(From,D,To),free(From),[]) :-
   action(push(From,D,To)),
   \+ step(From,D,To).
```

Encoding in B

Basic Static Causal Laws

```
caused([free(L)],neg(box_in(L))) :- location(L).

caused([free(L)],neg(sokoban_in(L))) :- location(L).

caused([sokoban_in(L)],neg(free(L))) :- location(L).

caused([sokoban_in(L)],neg(box(L))) :- location(L).

caused([sokoban_in(L1)],neg(sokoban_in(L2))) :-
            location(L1), location(L2), neq(L1,L2).

caused([box_in(L)],neg(free(L))) :- location(L).

caused([box_in(L)],neg(sokoban_in(L))) :- location(L).
```

Encoding in B

Static Causal Laws: reachability

```
caused([sokoban_in(A)],reachable(A)) :-
    location(A).

caused([reachable(B),free(C)],reachable(C)) :-
     location(B),location(C),
     neq(B,C),
     step(B,D,C),direction(D).
```

The results

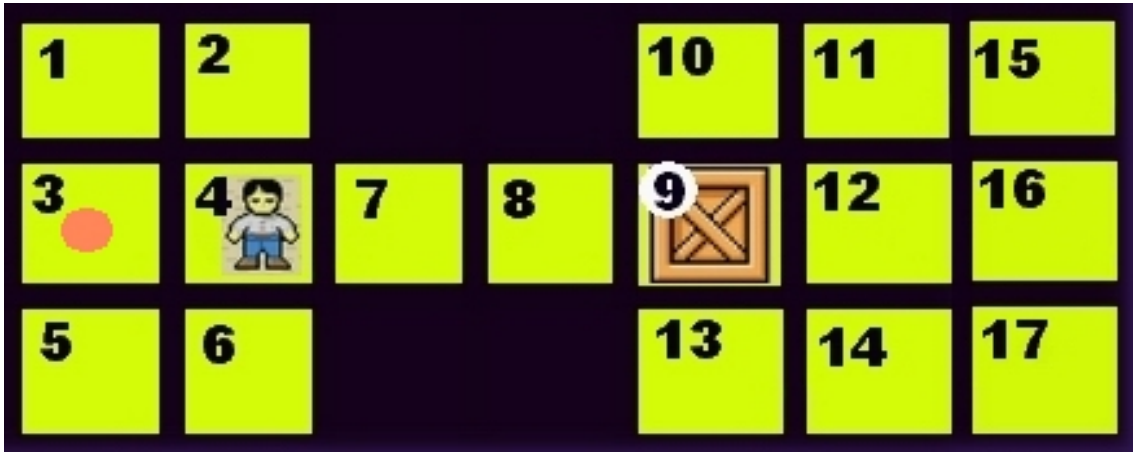
- This B encoding, compiled in ASP and run using clingo run rather fast on the proposed examples.
- We later discovered that the running time is comparable to that of the direct ASP solution of the Sokoban (also run using clingo)

The results

- This B encoding, compiled in ASP and run using clingo run rather fast on the proposed examples.
- We later discovered that the running time is comparable to that of the direct ASP solution of the Sokoban (also run using clingo)
- Unfortunately, the same did not holds for the CLP(FD) encoding (even if speed was not the real problem) which was our overall goal

Static Causal Laws as a (simple) constraint

`push(9,left,3)` is forbidden (12 is not reachable from 4).

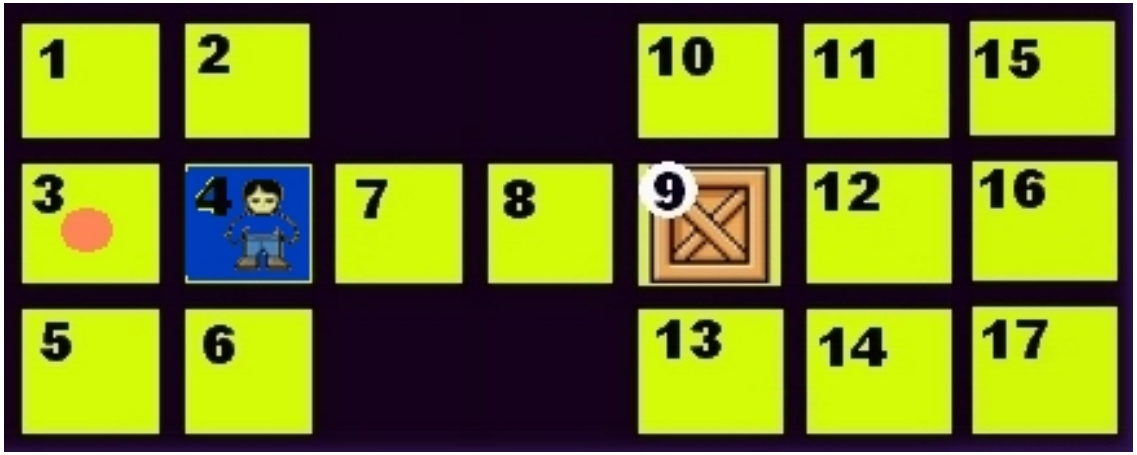



```
caused([sokoban_in(A)],reachable(A)) :-
    location(A).

caused([reachable(B),free(C)],reachable(C)) :-
     location(B),location(C),
     neq(B,C),
     step(B,D,C),direction(D).
```

Static Causal Laws as a (simple) constraint

`push(9,left,3)` is forbidden (12 is not reachable from 4).

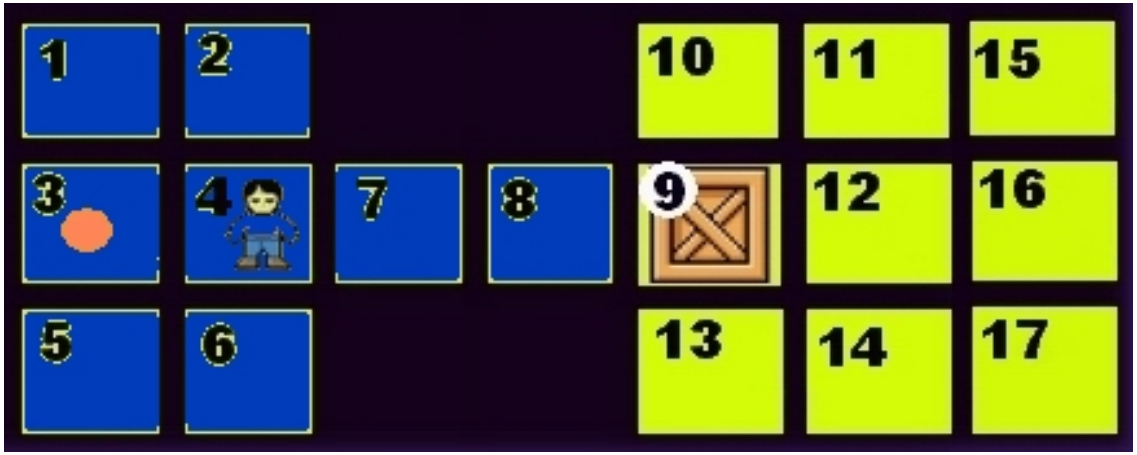



```
caused([sokoban_in(A)],reachable(A)) :-
    location(A).

caused([reachable(B),free(C)],reachable(C)) :-
     location(B),location(C),
     neq(B,C),
     step(B,D,C),direction(D).
```

Static Causal Laws as a (simple) constraint

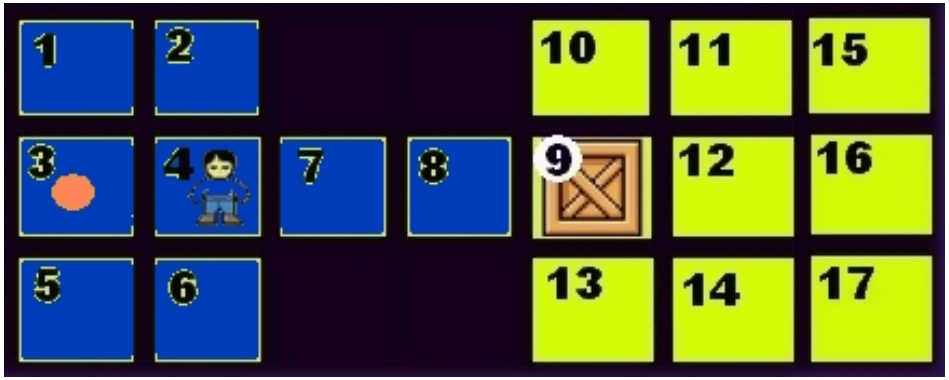
`push(9,left,3)` is forbidden (12 is not reachable from 4).




```
caused([sokoban_in(A)],reachable(A)) :-
    location(A).

caused([reachable(B),free(C)],reachable(C)) :-
     location(B),location(C),
     neq(B,C),
     step(B,D,C),direction(D).
```

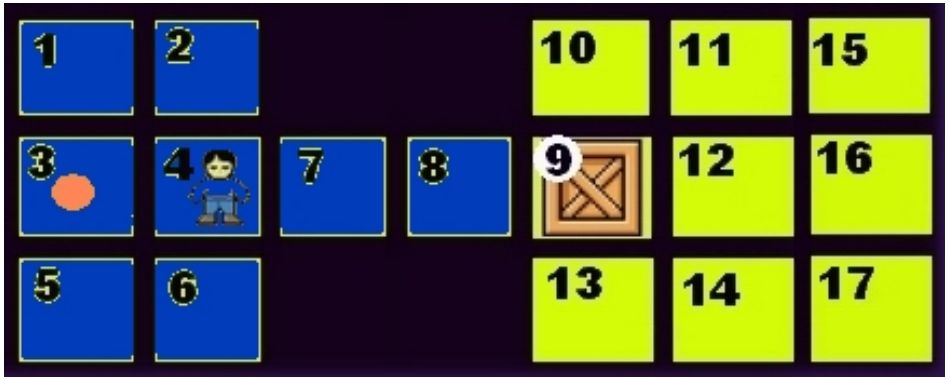
Static Causal Laws as a (simple) constraint




```
caused([reachable(B),free(C)],reachable(C)) :-
   location(B),location(C),neq(B,C),
   step(B,D,C),direction(D).
```

$$\texttt{reachable(11)} \wedge \underline{\texttt{free(12)}} \rightarrow \texttt{reachable(12)}.$$
$$\texttt{reachable(12)} \wedge \underline{\texttt{free(11)}} \rightarrow \texttt{reachable(11)}.$$

Static Causal Laws as a (simple) constraint

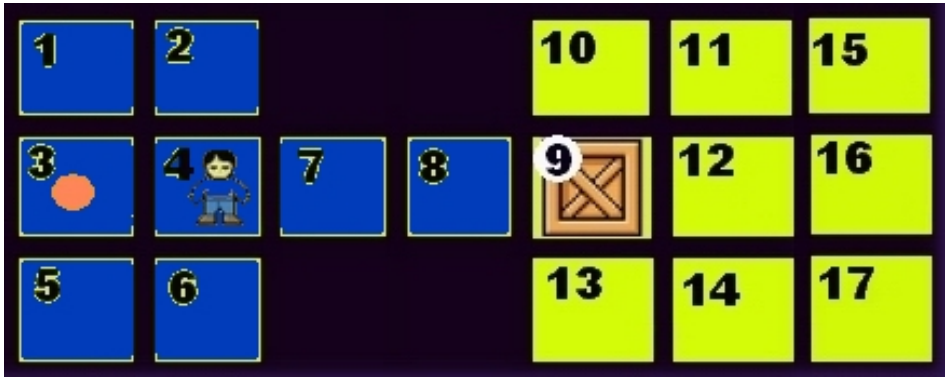



```
caused([reachable(B),free(C)],reachable(C)) :-
   location(B),location(C),neq(B,C),
   step(B,D,C),direction(D).
```

$$\texttt{reachable(11)} \wedge \underline{\texttt{free(12)}} \rightarrow \texttt{reachable(12)}.$$
$$\texttt{reachable(12)} \wedge \underline{\texttt{free(11)}} \rightarrow \texttt{reachable(11)}.$$

`reachable(11)` and `reachable(12)` both true is a solution (of the constraint).

Static Causal Laws as a (simple) constraint



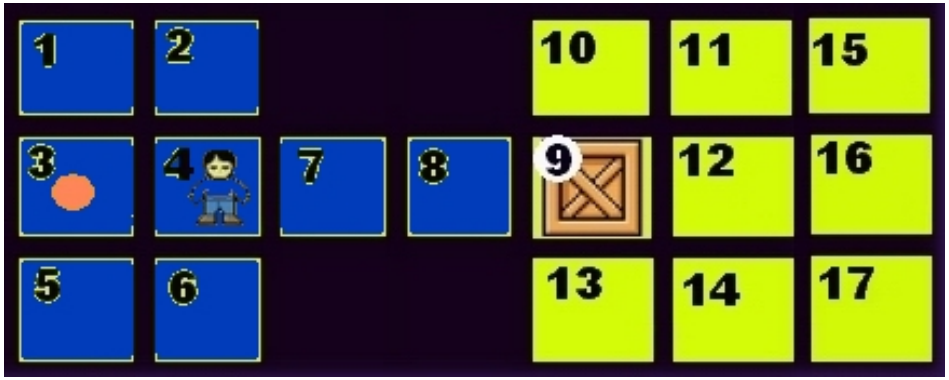

```
caused([reachable(B),free(C)],reachable(C)) :-
   location(B),location(C),neq(B,C),
   step(B,D,C),direction(D).
```

$$\texttt{reachable(11)} \wedge \underline{\texttt{free(12)}} \rightarrow \texttt{reachable(12)}.$$
$$\texttt{reachable(12)} \wedge \underline{\texttt{free(11)}} \rightarrow \texttt{reachable(11)}.$$

`reachable(11)` and `reachable(12)` both true is a solution (of the constraint).

This (loop) problem is correctly addressed by the ASP encoding.

Static Causal Laws as a (simple) constraint




```
caused([reachable(B),free(C)],reachable(C)) :-
   location(B),location(C),neq(B,C),
   step(B,D,C),direction(D).
```

$$\texttt{reachable(11)} \wedge \underline{\texttt{free(12)}} \rightarrow \texttt{reachable(12)}.$$
$$\texttt{reachable(12)} \wedge \underline{\texttt{free(11)}} \rightarrow \texttt{reachable(11)}.$$

`reachable(11)` and `reachable(12)` both true is a solution (of the constraint).

This (loop) problem is correctly addressed by the ASP encoding.

A correct constraint encoding would introduce too many constraints (making the CLP(FD) interpreter too slow).

Tabling

- Tabling has become a well-known and useful feature of many Prolog systems.
- The idea of tabling is to memorize answers to tabled subgoals and use the answers to resolve subsequent variant or subsumed subgoals.
- This idea resembles the dynamic programming idea of reusing solutions to overlapping sub-problems.

Tabling

- Tabling has become a well-known and useful feature of many Prolog systems.
- The idea of tabling is to memorize answers to tabled subgoals and use the answers to resolve subsequent variant or subsumed subgoals.
- This idea resembles the dynamic programming idea of reusing solutions to overlapping sub-problems.
- B-Prolog is a tabled Prolog system based on linear tabling, allows variant subgoals to share answers, and uses the local (lazy) strategy to return answers.

Tabling in B-Prolog

Fibonacci numbers

```
:-table fib/2.
fib(0, 1).
fib(1, 1).
fib(N, F):-
    N>1, N1 is N-1, N2 is N-2,
    fib(N1, F1),   fib(N2, F2),
    F is F1+F2.
```

Tabling in B-Prolog

Fibonacci numbers

```
:-table fib/2.
fib(0, 1).
fib(1, 1).
fib(N, F):-
    N>1, N1 is N-1, N2 is N-2,
    fib(N1, F1),   fib(N2, F2),
    F is F1+F2.
```

- Without tabling, the subgoal `fib(N,X)` would spawn 2^N subgoals, many of which are variants.

Tabling in B-Prolog

Fibonacci numbers

```
:-table fib/2.
fib(0, 1).
fib(1, 1).
fib(N, F):-
    N>1, N1 is N-1, N2 is N-2,
    fib(N1, F1),   fib(N2, F2),
    F is F1+F2.
```

- Without tabling, the subgoal `fib(N,X)` would spawn 2^N subgoals, many of which are variants.
- With tabling, the time complexity drops to linear since the same variant subgoal is resolved only once.

Tabling in B-Prolog

Modes

- B-Prolog allows Mode-directed tabling

```
:-table p(M1,...,Mn):C.
```

- `C` (optional), the *cardinality limit*, bounds the number of answers to be tabled for `p`,
- Each `Mi` is a mode:
  - `+` (input $\longrightarrow$ usually ground)
  - `-` (output $\longrightarrow$ usually a variable)
  - `min` or `max` (optimized $\longrightarrow$ output)
- Only one argument in a tabled predicate can have the mode `min` or `max`.

Tabling in B-Prolog

Shortest path (sp) in a weighted directed graph $X \xrightarrow{w} Y$

```
:-table sp(+,+,-,min).
sp(X,Y,[(X,Y)],W) :-
    edge(X,Y,W).
sp(X,Y,[(X,Z)|Path],W) :-
    edge(X,Z,W1),
    sp(Z,Y,Path,W2),
    W is W1+W2.
```

The predicate `sp(X,Y,Path,W)` states that `Path` is a path from `X` to `Y` with the smallest weight `W`.

Tabling in B-Prolog

Shortest path (sp) in a weighted directed graph $X \xrightarrow{w} Y$

```
:-table sp(+,+,-,min).
sp(X,Y,[(X,Y)],W) :-
    edge(X,Y,W).
sp(X,Y,[(X,Z)|Path],W) :-
    edge(X,Z,W1),
    sp(Z,Y,Path,W2),
    W is W1+W2.
```

The predicate `sp(X,Y,Path,W)` states that `Path` is a path from `X` to `Y` with the smallest weight `W`.
For each pair of nodes, only one (shortest) answer is tabled!

The Sokoban program

- Neng-Fa's implementation of the Sokoban program is based on the just seen tabled definition of shortest path
- The overall code is very short and simple (as one might expect from Prolog programming)
- A little (actually very little) domain knowledge is added

The Sokoban program

```
:-table plan_sokoban(+,+,-,min).
plan_sokoban(_SokobanLoc,BoxLocs,Plan,Len):-
    goal_reached(BoxLocs),!,   Plan=[],Len=0.
plan_sokoban(SokobanLoc,BoxLocs,
             [push(BoxLoc,Dir,DestLoc)|Plan],Len):-
    select(BoxLoc,BoxLocs,BoxLocs1),
    step(PrevNeibLoc,Dir,BoxLoc),
    \+ member(PrevNeibLoc,BoxLocs1),
    step(BoxLoc,Dir,NextNeibLoc),
    good_dest(NextNeibLoc,BoxLocs1),
    reachable_by_sokoban(SokobanLoc,PrevNeibLoc,BoxLocs),
    choose_dest(BoxLoc,NextNeibLoc,Dir,
                DestLoc,NewSokobanLoc,BoxLocs1),
    insert_ordered(DestLoc,BoxLocs1,NewBoxLocs),
    plan_sokoban(NewSokobanLoc,NewBoxLocs,Plan,Len1),
    Len is Len1+1.
```

The Sokoban program

```
:-table reachable_by_sokoban/3.
reachable_by_sokoban(Loc,Loc,_BoxLocs).
reachable_by_sokoban(Loc1,Loc2,BoxLocs):-
    step(Loc1,_,Loc3),
    \+ member(Loc3,BoxLocs),
    reachable_by_sokoban(Loc3,Loc2,BoxLocs).


choose_dest(Loc,NextLoc,_Dir,Dest,NewSokobanLoc,_BoxLocs):-
    Dest=NextLoc, NewSokobanLoc=BoxLoc.
choose_dest(Loc,NextLoc,Dir,Dest,NewSokobanLoc,BoxLocs):-
    step(NextLoc,Dir,NextNextLoc),
    good_dest(NextNextLoc,BoxLocs),
    choose_dest(NextLoc,NextNextLoc,Dir,
                Dest,NewSokobanLoc,BoxLocs).
```

The Sokoban program

Domain Knowledge

```
good_dest(Loc,BoxLocs):-
    \+ member(Loc,BoxLocs),
    (corner(Loc)->storage(Loc);true),
    foreach(BoxLoc in BoxLocs, \+ stuck(BoxLoc,Loc)).
:-table stuck/2.
stuck(X,Y):- (right(X,Y);right(Y,X)),
    (\+ storage(X); \+ storage(Y)),
    (\+ top(X,_), \+ top(Y,_);
     \+ top(_,X), \+ top(_,Y)),!.
stuck(X,Y):- (top(X,Y);top(Y,X)),
    (\+ storage(X); \+ storage(Y)),
    (\+ right(X,_), \+ rights(Y,_);
     \+ right(_,X), \+ right(_,Y)),!.
```

Two boxes constitute a deadlock if they are next to each other and both adjacent to a wall, unless both their locations are storage squares.

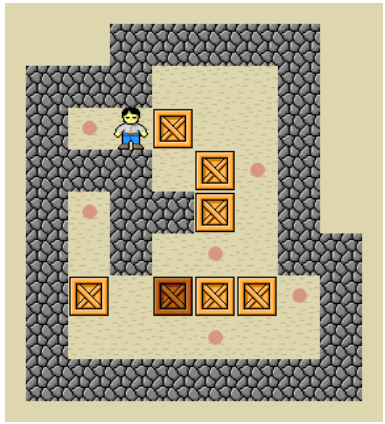
Competition results

CPU time, seconds

Instance	BPSolver	Clasp
1-sokoban-optimization-0-0.asp	0.58	0.06
13-sokoban-optimization-0-0.asp	0.06	0.74
18-sokoban-optimization-0-0.asp	0.00	9.80
20-sokoban-optimization-0-0.asp	33.57	13.24
24-sokoban-optimization-0-0.asp	2.66	3.52
27-sokoban-optimization-0-0.asp	0.78	1.16
29-sokoban-optimization-0-0.asp	0.78	2.92
33-sokoban-optimization-0-0.asp	1.96	26.74
37-sokoban-optimization-0-0.asp	0.38	8.52
4-sokoban-optimization-0-0.asp	Mem Out	0.62
43-sokoban-optimization-0-0.asp	Mem Out	35.67
45-sokoban-optimization-0-0.asp	Mem Out	9.30
47-sokoban-optimization-0-0.asp	Mem Out	18.66
5-sokoban-optimization-0-0.asp	0.00	0.16
9-sokoban-optimization-0-0.asp	0.00	2.12

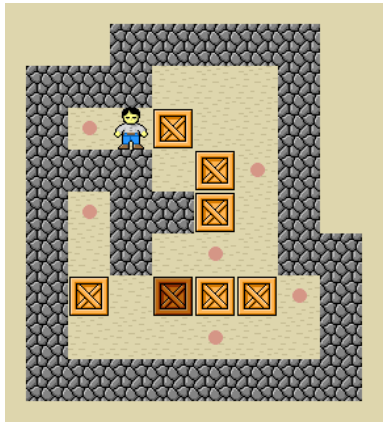
Conclusions

- We have played with the Sokoban game using several Logic Programming tools



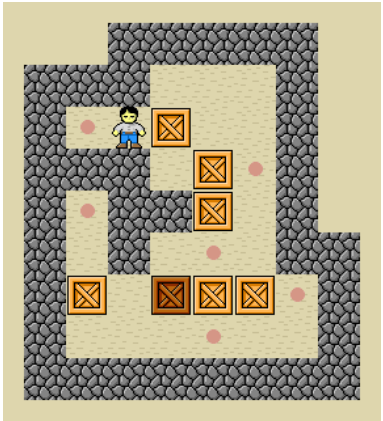
Conclusions

- We have played with the Sokoban game using several Logic Programming tools
- It was funny



Conclusions

- We have played with the Sokoban game using several Logic Programming tools
- It was funny
- Direct ASP (or B translated to ASP) works
- B interpreted by CLP(FD) does not work correctly (but we are now developing and exploiting a special reachability global constraint)



Conclusions

- We have played with the Sokoban game using several Logic Programming tools
- It was funny
- Direct ASP (or B translated to ASP) works
- B interpreted by CLP(FD) does not work correctly (but we are now developing and exploiting a special reachability global constraint)
- Tabled B-Prolog works (even if there are still some memory problems to cope with)
- Adding knowledge, of course, helps