`PuzzleTool`: an example of programming computation and deduction

Michael J. C. Gordon

University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue, Cambridge CB3 0FD, U.K.
Email: `mjcg@cl.cam.ac.uk` Web: `http://www.cl.cam.ac.uk/~mjcg`

Abstract. Systems that integrate user-programmable theorem proving with efficient algorithms for boolean formula manipulation are promising platforms for implementing special-purpose tools that combine computation and deduction. An example tool is presented in this paper in which theorem proving is used to compile a class of problems stated in terms of functions operating on sets of integers to boolean problems that can be solved using a BDD oracle. The boolean solutions obtained via BDD calculations are then converted by theorem proving to the high-level representation. Although the example is rather specialised, our goal is to illustrate methodological principles for programming tools whose operation requires embedded proof.

1 Background and motivation

There are already commercial Electronic Design Automation (EDA) tools[1] that have embedded model checkers. It seems likely that future tools will also require embedded theorem proving.

The PROSPER EU Esprit project[2] investigated embedding theorem proving inside both hardware and software applications. PROSPER focussed on architectural issues in linking systems [4]. The work reported here is narrower and concerns the programming of deduction and computation within a single system, HOL98, and describes:

- scripting combininations of theorem proving and symbolic computation;
- the 'logical flow' in compiling from an abstract representation to a boolean representation and lifting the boolean results back to the abstract level;
- a case study of a particular tool implementation.

In the next section we define the class of problems solved by our example tool. In the following sections the deductions necessary to solve problems in the class are explained and discussed.

[1] Examples are Avanti's clock domain and one-hot checkers (`www.avanticorp.com`) and 0-In search (`www.0in.com`).

[2] See `http://www.dcs.gla.ac.uk/prosper. PROSPER originated the ``plug-in'' and ``proof engine'' terminology used in some commercial products. See the web page www.prover.com of the PROSPER partner Prover Technologies.`

2 A class of puzzles generalising Peg Solitaire

Two example puzzles are first described, followed by a general setting in which they can be formulated. A third very simple example is given in Section 4.3 to illustrate the operation of our puzzle solving tool `PuzzleTool`.

2.1 Classical Peg Solitaire

Consider the following board

```
                -------------------------
                | xxxxx | xxxxx | xxxxx |
                -------------------------
                | xxxxx | xxxxx | xxxxx |
---------------------------------------------------------
| xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx |
---------------------------------------------------------
| xxxxx | xxxxx | xxxxx |       | xxxxx | xxxxx | xxxxx |
---------------------------------------------------------
| xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx | xxxxx |
---------------------------------------------------------
                | xxxxx | xxxxx | xxxxx |
                -------------------------
                | xxxxx | xxxxx | xxxxx |
                -------------------------
```

All the positions in the board, except the one in the middle, are occupied by pegs, denoted by `xxxxx`. A move consists of 'jumping' a peg over an adjacent peg in the same row or column into a hole, and removing the peg that was jumped over from the board (thereby reducing the number of pegs on the board by one). The puzzle is to find a sequence of moves, starting from the above configuration, to a configuration consisting of just one peg in the middle, i.e.:

```
                -------------------------
                |       |       |       |
                -------------------------
                |       |       |       |
---------------------------------------------------------
|       |       |       |       |       |       |       |
---------------------------------------------------------
|       |       |       | xxxxx |       |       |       |
---------------------------------------------------------
|       |       |       |       |       |       |       |
---------------------------------------------------------
                |       |       |       |
                -------------------------
                |       |       |       |
                -------------------------
```

Peg Solitaire can easily formulated as a state exploration problem by assigning a boolean variable to each board position, for example:

```
                -------------------------
                |  v00  |  v02  |  v03  |
                -------------------------
                |  v04  |  v05  |  v06  |
---------------------------------------------------------
|  v07  |  v08  |  v09  |  v10  |  v11  |  v12  |  v13  |
---------------------------------------------------------
|  v14  |  v15  |  v16  |  v17  |  v18  |  v19  |  v20  |
---------------------------------------------------------
|  v21  |  v22  |  v23  |  v24  |  v25  |  v26  |  v27  |
---------------------------------------------------------
                |  v28  |  v29  |  v30  |
                -------------------------
                |  v31  |  v32  |  v33  |
                -------------------------
```

The initial state is represented by

```
v01 ∧ v02 ∧ ... ∧ v16 ∧ ¬v17 ∧ v18 ∧ ... ∧ v33
```

and the final(goal) state by

```
¬v01 ∧ ¬v02 ∧ ... ∧ ¬v16 ∧ v17 ∧ ¬v18 ∧ ... ∧ ¬v33
```

The transition relation, say `R`, is then defined to be a disjunctions of terms, with one disjunct per possible move, so that `R((v01,...,v33),(v01',...,v33'))` is true if and only if there exists a move in state `(v01,...,v33)` that results in state `(v01',...,v33')`. Inspection of the board shows that there are 76 moves. The term for a move specifies a change to three variable and that the remaining 30 variables are unchanged. For example, the term

```
v05 ∧ v10 ∧ ¬v17 ∧ ¬v05' ∧ ¬v10' ∧ v17' ∧
(v01' = v01) ∧ (v02' = v02) ∧ (v03' = v03) ∧ (v04' = v04) ∧
(v06' = v06) ∧ (v07' = v07) ∧ (v08' = v08) ∧ (v09' = v09) ∧
(v11' = v11) ∧ (v12' = v12) ∧ (v13' = v13) ∧ (v14' = v14) ∧
(v15' = v15) ∧ (v16' = v16) ∧ (v18' = v18) ∧ (v19' = v19) ∧
(v20' = v20) ∧ (v21' = v21) ∧ (v22' = v22) ∧ (v23' = v23) ∧
(v24' = v24) ∧ (v25' = v25) ∧ (v26' = v26) ∧ (v27' = v27) ∧
(v28' = v28) ∧ (v29' = v29) ∧ (v30' = v30) ∧ (v31' = v31) ∧
(v32' = v32) ∧ (v33' = v33)
```

specifies that the peg at position `05` jump over the peg at position `10` into the centre hole (i.e. the one at position `17`), and all other positions remain unchanged.

Standard BDD methods can be used to compute a sequence of states starting with the initial state, ending with the final state, and such that adjacent elements in the sequance satisfy the transition relation `R`.

2.2 HexSolitaire

Consider now a variation of Peg Solitaire called HexSolitaire.

```
                _____
               /     \
         _____/ xxxxx \_____
        /     \       /     \
  _____/ xxxxx \_____/ xxxxx \_____
 /     \       /     \       /     \
/ xxxxx \_____/ xxxxx \_____/ xxxxx \
\       /     \       /     \       /
 \_____/ xxxxx \_____/ xxxxx \_____/
 /     \       /     \       /     \
/ xxxxx \_____/       \_____/ xxxxx \
\       /     \       /     \       /
 \_____/ xxxxx \_____/ xxxxx \_____/
 /     \       /     \       /     \
/ xxxxx \_____/ xxxxx \_____/ xxxxx \
\       /     \       /     \       /
 \_____/ xxxxx \_____/ xxxxx \_____/
       \       /     \       /
        \_____/ xxxxx \_____/
              \       /
               \_____/
```

====>

```
                _____
               /     \
         _____/       \_____
        /     \       /     \
  _____/       \_____/       \_____
 /     \       /     \       /     \
/       \_____/       \_____/       \
\       /     \       /     \       /
 \_____/       \_____/       \_____/
 /     \       /     \       /     \
/       \_____/ xxxxx \_____/       \
\       /     \       /     \       /
 \_____/       \_____/       \_____/
 /     \       /     \       /     \
/       \_____/       \_____/       \
\       /     \       /     \       /
 \_____/       \_____/       \_____/
       \       /     \       /
        \_____/       \_____/
              \       /
               \_____/
```

In HexSolitaire the moves are vertical or diagonal jumps, i.e. over abutting edges of hexagons. The six kinds of moves are illustrated below.

```
  _____
 /     \
/ xxxxx \_____
\       /     \
 \_____/ xxxxx \_____
       \       /     \
        \_____/       \
              \       /
               \_____/
```

===>

```
  _____
 /     \
/       \_____
\       /     \
 \_____/       \_____
       \       /     \
        \_____/ xxxxx \
              \       /
               \_____/
```

```
  _____                                _____
 /     \                              /     \
/       \_____                       / XXXXX \_____
\       /     \                      \       /     \
 \_____/ XXXXX \_____         ===>    \_____/       \_____
       \       /     \                      \       /     \
        \_____/ XXXXX \                      \_____/       \
              \       /                            \       /
               \_____/                              \_____/

                _____                                     _____
               /     \                                   /     \
         _____/ XXXXX \                            _____/       \
        /     \       /                           /     \       /
  _____/ XXXXX \_____/        ===>          _____/       \_____/
 /     \       /                           /     \       /
/       \_____/                           / XXXXX \_____/
\       /                                 \       /
 \_____/                                   \_____/

                _____                                     _____
               /     \                                   /     \
         _____/       \                            _____/ XXXXX \
        /     \       /                           /     \       /
  _____/ XXXXX \_____/        ===>          _____/       \_____/
 /     \       /                           /     \       /
/ XXXXX \_____/                           /       \_____/
\       /                                 \       /
 \_____/                                   \_____/

  _____                                     _____
 /     \                                   /     \
/ XXXXX \                                 /       \
\       /                                 \       /
 \_____/                                   \_____/
 /     \                                   /     \
/ XXXXX \                     ===>        /       \
\       /                                 \       /
 \_____/                                   \_____/
 /     \                                   /     \
/       \                                 / XXXXX \
\       /                                 \       /
 \_____/                                   \_____/

  _____                                     _____
 /     \                                   /     \
/       \                                 / XXXXX \
\       /                                 \       /
 \_____/                                   \_____/
 /     \                                   /     \
/ XXXXX \                     ===>        /       \
\       /                                 \       /
 \_____/                                   \_____/
 /     \                                   /     \
/ XXXXX \                                 /       \
\       /                                 \       /
 \_____/                                   \_____/
```

It turns out there is no solution to HexSolitaire!

3 `PuzzleTool`: a tool for puzzle designers

`PuzzleTool` is an aid for puzzle designers. It's input is a description of a board, an initial state, a goal state and a set of moves. The output is a solution if one exists. If there is no solution, then a set of alternative end states that do have solutions can be computed. The puzzle designer can then choose one of these computed reachable end states to create a solvable puzzle. In the rest of this paper it is shown how to program `PuzzleTool` in HOL98. Sections 6 and 7 contains a brief description of related work and some conclusions, respectively.

A board is represented by a finite set of holes located at positions `(x,y)` in a 2D grid (the co-ordinates `x` and `y` can be arbitrary integers). A state of the board is the subset of holes that have pegs in them.

For example, classical Peg Solitaire is represented as follows.[3]

```
SolitaireBoard =   {              (2,6); (3,6); (4,6);
                                  (2,5); (3,5); (4,5);
                    (0,4); (1,4); (2,4); (3,4); (4,4); (5,4); (6,4);
                    (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3);
                    (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2);
                                  (2,1); (3,1); (4,1);
                                  (2,0); (3,0); (4,0)}

SolitaireInitialState = SolitaireBoard DIFF {(3,3)}
SolitaireFinalState   = {(3,3)}
```

To specify the moves, eight auxiliary functions are defined which map a hole at `(x,y)` to the coordinates of the hole `n` units away in one of eight directions: north, south, east, west, north-west, north-east, south-west, south-east.

The following diagram illustrates the moves and auxiliary functions `N`, `S`, `E`, `W`, `NW`, `NE`, `SW` and `SE`. How these eight functions are used to define moves is explained later when the general transition function `Trans` is defined.

```
               N n (x,y)                      N  n (x,y) = (x, y+n)
                  |
 NW n (x,y)       |      NE n (x,y)           NW n (x,y) = (x-n,y+n)
          \       |     /                     NE n (x,y) = (x+n,y+n)
           \      |    /
            \     |   /
             \    |  /
              \   | /
  W n (x,y) ---(x,y)--- E n (x,y)             W  n (x,y) = (x-n,y)
              /   | \                         E  n (x,y) = (x+n,y)
             /    |  \
            /     |   \
           /      |    \
          /       |     \                     SW n (x,y) = (x-n,y-n)
 SW n (x,y)       |      SE n (x,y)           SE n (x,y) = (x+n,y-n)
                  |
               S n (x,y)                      S  n (x,y) = (x,  y-n)
```

A board and set of moves define a transition relation. For a particular puzzle a transition relation is obtained by applying `Trans`, defined below, to appropriate arguments.

```
⊢ Trans board moves (state, state') =
   ∃p q r ∈ board. (∃move ∈ moves. (q = move p) ∧ (r = move q)) ∧
                   (p IN state) ∧ (q IN state) ∧ ¬(r IN state) ∧
                   (state' = (state DIFF {p;q}) UNION {r})
```

[3] In HOL98 notation, a finite set is written $\{e_1;\ldots;e_n\}$, with semi-colon separating the elements. S_1 `DIFF` S_2 denotes the result of deleting all element in S_2 from S_1.

the arguments `board` and `moves` are, respectively, the set of holes and set of moves of the puzzle. A peg at `p` can jump over a peg at `q` into a hole at `r` in state `state` if they are in a line (i.e. `(q = move p)`$\wedge$`(r = move q)` for some move `move`), and if `p` and `q` have pegs in `state` (i.e. `(p IN state)`$\wedge$`(q IN state)`) and there is a hole at `r` in `state` (i.e. $\neg$`(r IN state)`). If `p` jumps over `q` into `r` then the successor state (i.e. `state'`) is obtained by removing `p` and `q` and adding `r` (i.e. `(state' = (state DIFF {p;q}) UNION {r})`, where S_1 `UNION` S_2 denotes the union of S_1 and S_2.

For classical Peg Solitaire the holes are adjacent, so `n=1`, and the set of moves are defined by `SolitaireMoves = {N 1; S 1; E 1; W 1}`.

The term `Trans SolitaireBoard SolitaireMoves` denotes the transition relation for Peg Solitaire, where `SolitaireBoard` and `SolitaireMove` are as defined above. The low-level representation consisting of the disjunction of 76 conjunctions can be derived by deduction from the definitions of `Trans`, `SolitaireBoard` and `SolitaireMove`. How this is done is explained in the next section.

HexSolitaire is most naturally specified with a board having holes in the same row two horizontal units apart (`n=2`), but holes in the same column one vertical unit apart (`n=1`).

```
HexSolitaireBoard =
{                           (2,8);
                     (1,7);        (3,7);
              (0,6);        (2,6);        (4,6);
                     (1,5);        (3,5);
              (0,4);        (2,4);        (4,4);
                     (1,3);        (3,3);
              (0,2);        (2,2);        (4,2);
                     (1,1);        (3,1);
                            (2,0)}

HexSolitaireInitialState = HexSolitaireBoard DIFF {(2,4)}
HexSolitaireFinalState   = {(2,4)}
HexSolitaireMoves        = {N 2; S 2; NE 1 ; SW 1 ; SE 1; NW 1}
```

In general, a puzzle is specified by four definitions: the board, the set of moves, the initial state and the final (or goal) state.

For `PuzzleTool` these are represented as HOL98 definitions of constants. For example, HexSolitaire is represented by the definitions of `HexSolitaireBoard`, `HexSolitaireInitialState`, `HexSolitaireFinalState` and `HexSolitaireMoves`.

`PuzzleTool` takes three definitions as arguments and returns a pair of functions. It has ML type `:thm * thm * thm -> (thm -> thm) * (unit -> thm)`. Evaluating

```
PuzzleTool(Board_def,Moves_def,InitialState_def)
```

returns two functions. The first is applied to the definition of a desired goal state to compute a solution. The second is applied to `()` to compute a theorem giving the set of reachable end states (an end state is a state from which no moves are possible). See Section 4.3 for an example.

4 How PuzzleTool works

First we describe abstractly how PuzzleTool works, then this account is specialised to the domain of puzzles and finally a very simple example is worked through.

4.1 Abstract view

Let B and R be polymorphic predicates representing a set of initial states and a transition relation, respectively.

$$B : \alpha \rightarrow \texttt{bool}$$
$$R : \alpha \times \alpha \rightarrow \texttt{bool}$$

Let `Reachable` $R\ B\ a$ if and only if a is reachable from some value satisfying B via a finite number of transitions of R. `Reachable` is a polymorphic function of type $(\alpha \times \alpha \rightarrow \texttt{bool}) \rightarrow (\alpha \rightarrow \texttt{bool}) \rightarrow \alpha \rightarrow \texttt{bool}$. We will use `Reachable` at two particular types: σ and τ.

Suppose we have a specific type σ, a specific predicate `B`, a specific transition relation `R` and a specific set of goal state specified by a predicate `Q`:

$$\texttt{B} : \sigma \rightarrow \texttt{bool}$$
$$\texttt{R} : \sigma \times \sigma \rightarrow \texttt{bool}$$
$$\texttt{Q} : \sigma \rightarrow \texttt{bool}$$

Think of σ as the type of states of puzzles (i.e. sets of those coordinates containing pegs). Suppose also that we have a way of encoding the subset of σ satisfying a predicate `D` as values of another type τ. In our application, values of type τ will be vectors of booleans and `D` characterises the states for a particular puzzle. The encoding is defined by functions `Abs` from σ to τ and `Rep` from τ to σ which satisfy the properties below.

$$\texttt{D} : \sigma \rightarrow \texttt{bool}$$
$$\texttt{Rep} : \sigma \rightarrow \tau$$
$$\texttt{Abs} : \tau \rightarrow \sigma$$

$$\vdash \forall \texttt{s}:\sigma.\ \texttt{D s = (Abs(Rep s) = s)}$$
$$\vdash \forall \texttt{t}:\tau.\ \texttt{Rep(Abs t) = t}$$
$$\vdash \forall \texttt{s}:\sigma.\ \texttt{B s} \Rightarrow \texttt{D s}$$
$$\vdash \forall \texttt{s s'}:\sigma.\ \texttt{D s} \wedge \texttt{R(s,s')} \Rightarrow \texttt{D s'}$$

Our first goal is to calculate a sequence of values of type σ starting from a value satisfying `B`, ending with a value satisfying `Q` and such that adjacent values in the sequence are related by `R`. In our application, such a sequence will be a puzzle solution.

We solve our first goal by calculating a sequence $\texttt{t}_1, \ldots, \texttt{t}_p$ of values of τ such that

$$\vdash \texttt{B(Abs t}_1\texttt{)},\ \vdash \texttt{R(Abs t}_1\texttt{,Abs t}_2\texttt{)}, \ldots,\ \vdash \texttt{R(Abs t}_{p-1}\texttt{,Abs t}_p\texttt{)},\ \vdash \texttt{Q(Abs t}_p\texttt{)}$$

In our application, t_1, ... , t_p will vectors of boolean constants computed using BDDs. Then by evaluating the terms `Abs` t_1, ... , `Abs` t_p one obtains the desired sequence of values of type σ. This is how solutions to puzzles are computed.

Suppose now we want to compute all the reachable states starting from `B` via transitions of `R` that satisfy some predicate `P`. In our example `P` characterises end states. We proceed by computing a term `W t` such that

```
⊢ Reachable (λ(t,t'). R(Abs t,Abs t')) (λt. B(Abs t)) t ∧ P(Abs t) = W t
```

In our example `W t` is obtained by using standard fixedpoint methods to compute the BDD of `Reachable (λ(t,t'). R(Abs t,Abs t')) (λt. B (Abs t)) t`, then conjoining this with the BDD of `P(Abs t)`, and then converting the resulting BDD to a term [5].

The following are routine theorems:

```
⊢ ∀D R B.
    (∀s. B s ⇒ D s) ∧ (∀s s'. D s ∧ R(s,s') ⇒ D s')
    ⇒
    (∀s. Reachable R B s ⇒ D s)

⊢ ∀D abs rep R B.
    (∀s. D s = (abs (rep s) = s)) ∧ (∀s. Reachable R B s ⇒ D s)
    ⇒
    ∀t. Reachable R B (abs t) =
         Reachable (λ(t,t'). R(abs t,abs t')) (λt. B (abs t)) t
```

Hence it follows by elementary reasoning from the properties on the previous page that:

```
⊢ ∀s. Reachable R B s ⇒ D s

⊢ ∀t. Reachable R B (Abs t) =
       Reachable (λ(t,t').R(Abs t,Abs t')) (λt.B (Abs t)) t
```

Simplifying the computed equation for `W t` with the second of these theorems yields:

```
⊢  ∀t. Reachable R B (Abs t) ∧ P(Abs t) = W t
```

hence by specialising `t` to `Rep s` and then generalising `s`

```
⊢  ∀s. Reachable R B (Abs(Rep s)) ∧ P(Abs(Rep s)) = W(Rep s)
```

hence

```
⊢  ∀s. Reachable R B s ∧ P s = D s ∧ W(Rep s)
```

Finally, by suitable logical manipulation one can prove for specific states $s_1,\cdots,s_q$ that:

$$\vdash \ \mathtt{W(Rep\ s)} = \mathtt{s} \in \{s_1,\cdots,s_q\}$$

See the following section for an example.

4.2 Concrete instantiation for `PuzzleTool`

The user of `PuzzleTool` starts by defining three constants to represent the board, the set of moves and the initial states. The names of these constants are chosen by the user, but let us assume here that they are `Board`, `Moves` and `Init` , respectively. Thus the inputs are something like:

```
val Board_def        = Define ‘Board = ... ‘;
val Moves_def        = Define ‘Moves = ... ‘;
val InitialState_def = Define ‘Init  = ... ‘;
```

The operation of `PuzzleTool` is then fully automatic and is split into two phases.

Phase 1 computes a bespoke theory of the puzzle to be analysed and theorems that relate the high level specifications to a boolean encoding. These are then packaged into two functions which, when invoked, perform BDD analysis. Phase 1 is invoked by evaluating something like:

```
val (SolveFun, EndStatesFun) =
 PuzzleTool(Board_def,Moves_def,InitialState_def)
```

Phase 2 uses the HOL98 library `HolBddLib` [6], to compute a solution (`SolveFun`) and/or the set of reachable end states (`EndStatesFun`). If a solution is computed then the user must first define the desired goal state by something like:

```
val Final_def = Define ‘Final = ... ‘;
```

and then apply `SolveFun` to `Final_def`.

The theorems proved in Phase 1 are used by the functions returned by `PuzzleTool` both to encode the problem for the BDD-based calculation in Phase 2 and also to lift the solutions found in Phase 2 to the original representation in terms of sets of holes.

The ML program implementing `PuzzleTool` starts by making a number of definitions derived from the arguments it is invoked with. First some predicates are defined:

```
⊢ BoardInit  state = (state = Init)
⊢ BoardDom   state = (state SUBSET Board)
```

where S_1 `SUBSET` S_2 if and only if S_1 is included in S_2.

Next a transition relation for the specific puzzle is defined by applying `Trans` to `Board` and `Moves`.

```
⊢ BoardTrans = Trans Board Moves
```

and it is proved that

```
⊢ ∀state state'.
    BoardDom state ∧ BoardTrans(state,state') ⇒ BoardDom state'
```

abstraction and representation functions `BoardAbs` and `BoardRep` are then defined. `BoardAbs` maps states to tuples of booleans (boolean vectors) and `BoardRep` is a partial inverse. A boolean vector `(b1,...,bn)` represents a set of states via an enumeration of the holes in the board. Hole `p` is in the set represented by `(b1,...,bn)` if and only if `bi` is true, where `i` is the position of `p` in the enumeration.

The enumeration of holes is specified by defining a non-repeating list of the board positions:

```
⊢ BoardList = [...]
```

where the elements of the list are computed from `Board`. The following general functions are used to define the abstraction and representation:

```
⊢ REP pl state = MAP (λp. p IN state) pl
```

```
⊢ (ABS [] [] = {})
  ∧
  (ABS (p::pl) (b::bl) =
    if b then p INSERT ABS pl bl else ABS pl bl)
```

where `MAP` maps a function over a list, `::` (a infix) is the list 'cons' function and `INSERT` (an infix) inserts an element into a set. To convert between tuples and lists, functions dependent on the size of the puzzle are automatically defined:

```
⊢ BoardL2T [b1;...;bn] = (b1,...,bn)
⊢ BoardT2L (b1,...,bn) = [b1;...;bn]
```

Using these, `PuzzleTool` defines

```
⊢ BoardAbs bv    = ABS BoardList (BoardT2L bv)
⊢ BoardRep state = BoardL2T(REP BoardList state)
```

The following theorems are then proved:

```
⊢ ∀state. BoardDom state = (BoardAbs(BoardRep state) = state)
⊢ ∀bv. BoardRep(BoardAbs bv) = bv
⊢ ∀state. BoardInit state ⇒ BoardDom state
```

The variable `bv` here and below is so named because it ranges over boolean vectors. Next it is proved that:

```
⊢ ∀state. Reachable BoardTrans BoardInit state ⇒ BoardDom state
```

and hence

```
⊢  Reachable BoardTrans BoardInit (BoardAbs(b1,...,bn)) =
     Reachable
      (λ((b1,...,bn),(b1',...,bn')).
        BoardTrans(BoardAbs(b1,...,bn),BoardAbs(b1',...,bn')))
      (λ(b1,...,bn). BoardInit(BoardAbs(b1,...,bn)))
      (b1,...,bn)
```

In order to evaluate the right hand side of this equation using BDDs, the terms `BoardTrans(BoardAbs(b1,...,bn),BoardAbs(b1',...,bn'))` and also the term `BoardInit(BoardAbs(b1,...,bn))` are rewritten using derived equations to get theorems of the form:

```
⊢  BoardTrans(BoardAbs(b1,...,bn),BoardAbs(b1',...,bn')) = ...
⊢  BoardInit(BoardAbs(b1,...,bn)) = ...
```

where the occurrences of "..." on the right hand side are boolean formulae suitable for direct representation as BDDs (see Section 4.3 for examples).

Suitable equations must be assumed or proved that enable the HOL98 rewriting tools[4] to automatically derive theorems of the form shown above. In the current implementation some of these equations are proved as general theorems that can be instantiated, some are proved on-the-fly each time the tool is invoked, and some are even assumed as axioms (Section 7 discusses this).

Phase 2 now takes over. To search for a solution a theorem:

```
⊢  BoardFinal(BoardAbs(b1,...,bn)) = ...
```

is also derived. If a solution exists, then standard state enumeration methods in `HolBddLib` can compute a sequence of theorems:

$$
\begin{array}{ll}
\vdash & \texttt{BoardInit(BoardAbs(}\mathtt{c}_{11}\texttt{,...,}\mathtt{c}_{1n}\texttt{))} \\
\vdash & \texttt{BoardTrans(BoardAbs(}\mathtt{c}_{11}\texttt{,...,}\mathtt{c}_{1n}\texttt{),BoardAbs(...))} \\
 & \vdots \\
\vdash & \texttt{BoardTrans(BoardAbs(...),BoardAbs(}\mathtt{c}_{m1}\texttt{,,...,}\mathtt{c}_{mn}\texttt{))} \\
\vdash & \texttt{BoardFinal(BoardAbs(}\mathtt{c}_{m1}\texttt{,...,}\mathtt{c}_{mn}\texttt{))}
\end{array}
$$

where ($\mathtt{c}_{11}$,...,$\mathtt{c}_{1n}$), ... , ($\mathtt{c}_{m1}$,...,$\mathtt{c}_{mn}$) are vectors of the boolean constants `T` (true) and `F` (false). Evaluating `BoardAbs(`$\mathtt{c}_{11}$`,...,`$\mathtt{c}_{1n}$`)`, ... , `BoardAbs(`$\mathtt{c}_{m1}$`,...,`$\mathtt{c}_{mn}$`)` using `computeLib` [2] yields a sequence of states representing a possible solution.

To compute the reachable end states, standard `HolBddLib` methods can compute the BDD representing the term

```
Reachable BoardTrans BoardInit (BoardAbs(b1,...,bn))
```

An end state is defined to be a state from which there are no transitions. The general definition is:

```
⊢ ∀R state. End R state = ¬∃state'. R(state,state')
```

[4] These tools are in the HOL98 libraries `simpLib` and `computeLib`.

Instantiating `R` to `(`λ`(bv,bv'). BoardTrans(BoardAbs bv,boardAbs bv'))`, `bv` to `(b1,...,bn)` and then simplifying yields a theorem of the form

```
⊢ End (λ(bv,bv'). BoardTrans(BoardAbs bv,BoardAbs bv')) (b1,...,bn) = ...
```

`HolBddLib` is then used to compute the BDD of the right hand side of this equations, and hence the BDD representing the left hand side. This BDD can then be combined with the BDD representing the end states and the result converted to a theorem of the form:

```
⊢  Reachable BoardTrans BoardInit (BoardAbs(b1,...,bn)) ∧
    End BoardTrans (BoardAbs(b1,...,bn)) = ...
```

where the right hand side "`...`" is the term obtained from the BDD. Rewriting this to DNF and also using the equations:

```
⊢ b = (b = T)
⊢ ¬b = (b = F)
⊢ (b1 = c1) ∧ (b2 = c2) == ((b1,b2) = (c1,c2))
```

yields a theorem of the form

```
⊢  ∀b1 ... bn
     Reachable BoardTrans BoardInit (BoardAbs(b1,...,bn)) ∧
      End BoardTrans (BoardAbs(b1,...,bn)) =
      ((b1,...,bn) = (c11,...,c1n))
      ∨
      ⋮
      ∨
      ((b1,...,bn) = (cp1,...,cpn))
```

where each constant $\mathtt{c}_{jk}$ is `T` or `F`. This is equivalent to

```
⊢  ∀bv.
     Reachable BoardTrans BoardInit (BoardAbs bv) ∧
      End BoardTrans (BoardAbs bv) =
      (bv = (c11,...,c1n)) ∨ ··· ∨ (bv = (cp1,...,cpn))
```

i.e. quantification of `n` boolean variables is replaced by a single quantification over a tuple. Instantiating `bv` to `BoardRep state` and then generalising yields:

```
⊢  ∀state.
     Reachable BoardTrans BoardInit (BoardAbs(BoardRep state)) ∧
      End BoardTrans (BoardAbs(BoardRep state)) =
      (BoardRep state = (c11,...,c1n))
      ∨
      ⋮
      ∨
      (BoardRep state = (cp1,...,cpn))
```

The following is a theorem

```
⊢ ∀abs rep.
     (∀bv. rep(abs bv) = bv)
     ⇒
     ∀s bv. (rep s = bv) = (abs(rep s) = abs bv)
```

hence as `⊢ ∀bv. BoardRep(BoardAbs bv) = bv`

```
⊢  ∀state bv.
      (BoardRep state = bv) = (BoardAbs(BoardRep state) = BoardAbs bv)
```

hence

```
⊢  ∀state.
      Reachable BoardTrans BoardInit (BoardAbs(BoardRep state)) ∧
       End BoardTrans (BoardAbs(BoardRep state)) =
       (BoardAbs(BoardRep state) = BoardAbs(c_11,...,c_1n))
       ∨
       ⋮
       ∨
       (BoardAbs(BoardRep state) = BoardAbs(c_p1,...,c_pn))
```

using

```
⊢ ∀state. BoardDom state = (BoardAbs(BoardRep state) = state)
⊢ ∀state. Reachable BoardTrans BoardInit state ⇒ BoardDom state
```

the theorem above simplifies to

```
⊢  ∀state.
      Reachable BoardTrans BoardInit state ∧ End BoardTrans state =
        BoardDom state
        ∧
        ((state = BoardAbs(c_11,...,c_1n))
         ∨
         ⋮
         ∨
         (state = BoardAbs(c_p1,...,c_pn)))
```

which can be rewritten to

```
⊢  ∀state.
      Reachable BoardTrans BoardInit state ∧ End BoardTrans state =
        state SUBSET Board
        ∧
        state IN {BoardAbs(c_11,...,c_1n)); ... ; BoardAbs(c_p1,...,c_pn)}
```

which shows the set of reachable end states to be:

```
{BoardAbs(c_11,...,c_1n) ; ... ; BoardAbs(c_p1,...,c_pn)}
```

Each member of this set can be evaluated to state (i.e. a set of hole positions).

4.3 Example: puzzle `Triv`

A very simple puzzle called `Triv` that will be used to illustrate the operation of `PuzzleTool`.

```
                                                ---------          ---------
TrivBoard = {(0,4);                             |       |          | XXXXX |
             (0,3);                             ---------          ---------
             (0,2);                             | XXXXX |          |       |
             (0,1);                             ---------          ---------
             (0,0)}                             |       |   ===>   |       |
                                                ---------          ---------
TrivMoves = {N 1; S 1}                          | XXXXX |          |       |
TrivInitialState = {(0,0);(0,1);(0,3)}          ---------          ---------
TrivFinalState = {(0,4)}                        | XXXXX |          |       |
                                                ---------          ---------
```

The first step in solving `Triv` is to define constants representing the puzzle:

```
val TrivBoard_def        = Define‘TrivBoard={(0,4);(0,3);(0,2);(0,1);(0,0)}‘
val TrivMoves_def        = Define‘TrivMoves={N 1;S 1}‘
val TrivInitialState_def = Define‘TrivInitialState={(0,0);(0,1);(0,3)}‘
val TrivFinalState_def   = Define‘TrivFinalState={(0,4)}‘
```

The function `puzzleTool` performs the Phase 1 analysis and returns a pair of functions for doing the Phase 2 BDD analysis. The first function computes solutions and the second one computes reachable end states. Here is a complete session to solve and analyse `Triv` (some system output deleted).

```
- val (TrivSolveFn,TrivEndStatesFn) =
   PuzzleTool(TrivBoard_def,
              TrivMoves_def,
              TrivInitialState_def);

> val TrivSolveFn = fn : thm -> thm
  val TrivEndStatesFn = fn : unit -> thm

- TrivEndStatesFn();
> val it =
    |- Reachable TrivTrans TrivInit s /\ End TrivTrans s =
       s SUBSET TrivBoard /\ s IN {{(0,4)}; {(0,1)}} : thm

- val TrivFinalState_def = Define ‘TrivFinalState = {(0,4)}‘;
> val TrivFinalState_def = |- TrivFinalState = {(0,4)} : thm

- TrivSolveFn TrivFinalState_def;
> val it =
    |- TrivInit {(0,3); (0,1); (0,0)} /\
       TrivTrans ({(0,3); (0,1); (0,0)},{(0,3); (0,2)}) /\
       TrivTrans ({(0,3); (0,2)},{(0,4)}) /\ TrivFinal {(0,4)} : thm
```

Thus invoking PuzzleTool on Triv yields the following trace:

```
[{(0,0);(0,1);(0,3)}; {(0,2);(0,3)}; {(0,4)}]
```

and the set {{(0,4)}; {(0,1)}} of reachable end states.

The results of invoking PuzzleTool on the two solitaire games can be found in the appendices at the end of this paper.

Phase 1 begins with a number of definitions being made.

```
⊢ TrivInit state = (state = {(0,0); (0,1); (0,3)})

⊢ TrivAbs (b0,b1,b2,b3,b4) =
   ABS [(0,4); (0,3); (0,2); (0,1); (0,0)] [b0; b1; b2; b3; b4]

⊢ TrivT2L [b0; b1; b2; b3; b4] = (b0,b1,b2,b3,b4)

⊢ TrivRep s = TrivT2L (REP [(0,4); (0,3); (0,2); (0,1); (0,0)] s)

⊢ TrivDom s = s SUBSET {(0,0); (0,1); (0,3)}

⊢ TrivTrans (state,state') = Trans TrivBoard TrivMoves (state,state')
```

From these definitions, some axioms[5] and pre-proved general theorems, the boolean representation of the puzzle is computed:

```
⊢ TrivInit(TrivAbs(b0,b1,b2,b3,b4)) = ¬b0 ∧ b1 ∧ ¬b2 ∧ b3 ∧ b4

⊢ TrivTrans(TrivAbs(b0,b1,b2,b3,b4),TrivAbs(b0',b1',b2',b3',b4')) =
    (b2 ∧ b1 ∧ ¬b0 ∧ b0' ∧ ¬b1' ∧ ¬b2' ∧ (b3' = b3) ∧ (b4' = b4) ∨
     b3 ∧ b2 ∧ ¬b1 ∧ (b0' = b0) ∧ b1' ∧ ¬b2' ∧ ¬b3' ∧ (b4' = b4) ∨
     b4 ∧ b3 ∧ ¬b2 ∧ (b0' = b0) ∧ (b1' = b1) ∧ b2' ∧ ¬b3' ∧ ¬b4') ∨
    b0 ∧ b1 ∧ ¬b2 ∧ ¬b0' ∧ ¬b1' ∧ b2' ∧ (b3' = b3) ∧ (b4' = b4) ∨
    b1 ∧ b2 ∧ ¬b3 ∧ (b0' = b0) ∧ ¬b1' ∧ ¬b2' ∧ b3' ∧ (b4' = b4) ∨
    b2 ∧ b3 ∧ ¬b4 ∧ (b0' = b0) ∧ (b1' = b1) ∧ ¬b2' ∧ ¬b3' ∧ b4'
```

These 'scalarised' equations are built into the functions returned by PuzzleTool.

Phase 2 can now start. The BDD state exploration tools in HOL98's HolBddLib can automatically compute a solution, given a goal state:

```
val FinalState_def = Define ‘TrivFinalState = {(0,4)}‘;
```

from which the function TrivSolveFn returned by PuzzleTool deduces:

```
⊢ TrivFinal (TrivAbs(b0,b1,b2,b3,b4)) = b0 ∧ ¬b1 ∧ ¬b2 ∧ ¬b3 ∧ ¬b4
```

and then uses HolBddLib to compute:

[5] See Section 7 for a discussion of assuming axioms versus proving theorems.

```
⊢ TrivInit(TrivAbs(F,T,F,T,T))
⊢ TrivBoardTrans(TrivAbs(F,T,F,T,T),TrivAbs(F,T,T,F,F))
⊢ TrivBoardTrans(TrivAbs(F,T,T,F,F),TrivAbs(T,F,F,F,F))
⊢ TrivFinal(TrivAbs(T,F,F,F,F))
```

which after evaluation and conjunction yield:

```
⊢ TrivInit {(0,3); (0,1); (0,0)} ∧
  TrivTrans ({(0,3); (0,1); (0,0)},{(0,3); (0,2)}) ∧
  TrivTrans ({(0,3); (0,2)},{(0,4)}) ∧
  TrivFinal {(0,4)}
```

To compute the reachable end states the following is derived purely inside HOL98.

```
⊢ End
   (λ(bv,bv'). TrivTrans(TrivAbs bv,TrivAbs bv'))
   (b0,b1,b2,b3,b4) =
    ((∀b1' b2'. ¬b2 ∨ ¬b1 ∨ b0 ∨ b1' ∨ b2') ∧
     (∀b2' b3'. ¬b3 ∨ ¬b2 ∨ b1 ∨ b2' ∨ b3') ∧
     ∀b3' b4'. ¬b4 ∨ ¬b3 ∨ b2 ∨ b3' ∨ b4') ∧
    (∀b0' b1'. ¬b0 ∨ ¬b1 ∨ b2 ∨ b0' ∨ b1') ∧
    (∀b1' b2'. ¬b1 ∨ ¬b2 ∨ b3 ∨ b1' ∨ b2') ∧
    ∀b2' b3'. ¬b2 ∨ ¬b3 ∨ b4 ∨ b2' ∨ b3'
```

The BDD of this is computed. `HolBddLib` is then invoked to compute the BDD of the set of all reachable states and the two BDDs are combined and the resulting BDD converted to a term to obtain:

```
⊢ Reachable TrivTrans TrivInit (TrivAbs(b0,b1,b2,b3,b4)) ∧
   End TrivTrans (TrivAbs(b0,b1,b2,b3,b4)) =
    b0 ∧ ¬b1 ∧ ¬b2 ∧ ¬b3 ∧ ¬b4 ∨ ¬b0 ∧ ¬b1 ∧ ¬b2 ∧ b3 ∧ ¬b4
```

from this it follows by deduction:

```
⊢ ∀b0 b1 b2 b3 b4.
    Reachable TrivTrans TrivInit (TrivAbs(b0,b1,b2,b3,b4)) ∧
    End TrivTrans (TrivAbs(b0,b1,b2,b3,b4)) =
     ((b0,b1,b2,b3,b4) = (T,F,F,F,F)) ∨ ((b0,b1,b2,b3,b4) = (F,F,F,T,F))
```

hence:

```
⊢ Reachable TrivTrans TrivInit (TrivAbs(TrivRep s)) ∧
   End TrivTrans (TrivAbs(TrivRep s)) =
    (TrivRep s = (T,F,F,F,F)) ∨ (TrivRep s = (F,F,F,T,F))
```

and then using theorems relating `TrivDom`, `TrivRep` and `TrivAbs`, and also the easily proved theorem `⊢ ∀s. Reachable TrivTrans TrivInit s ⇒ TrivDom s`

```
⊢ Reachable TrivTrans TrivInit s ∧ End TrivTrans s =
    TrivDom s ∧ ((s = {(0,4)}) ∨ (s = {(0,1)}))
```

and hence

```
⊢ Reachable TrivTrans TrivInit s ∧ End TrivTrans s =
    s SUBSET TrivBoard ∧ s IN {{(0,4)}; {(0,1)}}
```

5 Computation and deduction

The flow from input to output described in the previous section is programmed in the HOL98 metalanguage, Standard ML. `PuzzleTool` can be viewed logically as two derived inference rules in higher order logic. The rule corresponding to computing solutions is:

$$\frac{\vdash \texttt{Board = ...} \quad \vdash \texttt{Moves = ...} \quad \vdash \texttt{Init = ...} \quad \vdash \texttt{Final = ...}}{\begin{array}{l}\vdash \texttt{state}_1 \texttt{ IN Init} \\ \quad \wedge \\ \quad \texttt{BoardTrans(state}_1\texttt{, ...)} \wedge \quad \ldots \quad \wedge \texttt{ BoardTrans(..., state}_n\texttt{)} \\ \quad \wedge \\ \quad \texttt{state}_n \texttt{ IN Final}\end{array}}$$

the rule corresponding to computing all reachable end states is:

$$\frac{\vdash \texttt{Board = ...} \quad \vdash \texttt{Moves = ...} \quad \vdash \texttt{Init = ...}}{\begin{array}{l}\vdash \forall \texttt{state.} \\ \quad \texttt{Reachable BoardTrans BoardInit state} \wedge \texttt{ End BoardTrans state =} \\ \quad\ \texttt{state SUBSET Board} \wedge \texttt{ state IN } \{ \ldots \}\end{array}}$$

Puzzles are not critical systems, so the high assurance provided by the LCF-style proof that is generated is of little value. It is possible that other tools implemented using a similar methodology might gain value from this assurance[6], but the point we want to make in this paper is that even if assurance of soundness is not needed, the implementation style used for `PuzzleTool` is a powerful way of programming combinations of computation and deduction.

The conventional way to implement special purpose tools like `PuzzleTool` would be to either program them directly using a BDD package such as MuDDy [9] or CUDD [15], or to use a model checker like SMV or SPIN, which provide high level modelling and property languages and tools for checking. The advantage of the approach described here is that the implementer can develop a model within the expressive language of higher order logic and then implement customised functions to efficiently perform exactly the desired computations. The theorem proving infrastructure of tactics and simplifiers applied to a general logic becomes available for compiling problems into the domain of specialized oracles like BDD and SAT.

6 Related work

The seminal paper on connecting theorem proving and model checking is *An Integration of Model-Checking with Automated Proof Checking* by Rajan, Shankar

[6] `PuzzleTool` creates a theory each time it is invoked which contain various intermediate theorems. It is possible another application might find such theories a useful result.

and Srivas [13] in which PVS [12] is extended to support symbolic model checking of properties stated in the μ-calculus via a link to an external ROBDD-based μ-calculus checker. Model checking is invoked from PVS via a command that translates higher order logic goals into boolean formulae that can be printed into a format suitable for input to the external tool. The translation is not by programmed proof, as here, but by Lisp code that operates on the internal representation of PVS formulae. The PVS approach delivers a powerful shrink-wrapped checker, our approach is complementary: it provides a scripting framework for the user to implement bespoke tools.

PVS adds model checking to a theorem proving platform. The dual is to add theorem proving to a model checker. This has been done by McMillan in Cadence SMV [3], which provides problem decomposition commands that split verification goals into components small enough for model checking. The decomposition is based on deductive rules, for example for compositional refinement [10], and implemented by light-weight theorem proving. SMV doesn't provide a user-programmable scripting facility, rather each new deduction method is hardwired into the system. It may be possible to use the approach described in this paper to program the kind of algorithms that SMV builds-in as derived rules, but more work is needed to investigate this.

Carl Seger's Voss system [14] and its descendants have influenced and inspired the work described here. Voss consists of a lazy ML-like functional language, called FL, with BDDs as a built-in datatype. Quantified Boolean formulae can be input and are parsed to BDDs. Algorithms for model checking are easily programmed. Joyce and Seger interfaced an early HOL system (HOL88) to Voss and in a pioneering paper showed how to verify complex systems by a combination of theorem proving deduction and symbolic trajectory evaluation (STE) [8]. The HOL-Voss system integrates HOL deduction with BDD computations. BDD tools are programmed in FL and can then be invoked by HOL-Voss tactics, which can make external calls into the Voss system, passing subgoals via a translation between the HOL and Voss term representations. After Seger moved to Intel, the development of Voss evolved into a system called Forte that "is an LCF-style implementation of a higher-order classical logic"[1] and "seamlessly integrates several types of model-checking engines with lightweight theorem proving and extensive debugging capabilities, creating a productive high-capacity formal verification environment". Only partial details of this are in the public domain [11, 1], but a key idea is that FL is used both as a specification language and as an LCF-style metalanguage – i.e. there is no separation, as there is in LCF-like systems, between object language (higher order logic) and metalanguage (Standard ML). One difference between Voss/Forte and HOL98/`HolBddLib` is that the latter allows BDD programming in ML to be intimately mixed with the use of power tools from the HOL system. Theorem proving tools (e.g. simplifiers) can be directly applied to terms to transform and optimise them for BDD purposes.

A different approach is to implement BDDs inside the HOL system without making use of an external oracle. John Harrison has done this [7] and shown that the BDD algorithms provide a way of implementing tautology-checking that is significantly better than the methods previously used in HOL. He found, however, that performance was about a thousand times slower than with a BDD engine implemented in C.[7] By reimplementing some of HOL's primitive rules, performance could be improved by around ten times. Harrison only provided data for boolean equivalence checking. The approach in this paper aims to get near the performance of C-based model checking (by using a BDD package implemented in C), whilst remaining within the spirit of the fully-expansive LCF style. Harrison's work is 'logically purer' than ours, but less efficient. The tradeoff betweeen logical security and efficiency depends on the application, but Harrison's experiments on an internal implementation of BDDs provides a very interesting point between pure fully-expansive theorem proving and the use of external oracle.

7 Conclusions

The implementation of efficient special purpose tools in a general theorem prover like HOL98 is practicable. The implementer can produce a quick prototype using 'brute force' methods and by assuming general results as axioms before going to the effort of proving them. This can establish proof of concept.

The tool can then be tuned by optimising deductions using standard theorem proving and verification methods. The assurance of soundness can be improved by checking the assumed theorems for each problem 'on-the-fly' by symbolic execution, or by proving general theorems that can be instantiated. How much optimisation and assurance enhancement to do requires a cost/benefit analysis.

The needs of a tool implementation platform are wider than the needs of a pure proof assistant. Scripting and encapsulation mechanisms are necessary, and access to external oracles (like BDD and SAT engines) are essential for some applications. There is potential for synergy between the automatic theorem proving, proof assistant and verification tool communities. This paper has explored widening the applicability of a proof assistant, HOL98, to be a tool implementation platform. In the long term it might be worthwhile designing such a platform from scratch, but in the short to medium term there is plenty of potential left in existing programmable proof assistants.

8 Acknowledgements

Thanks to members of the Automated Reasoning Group at Cambridge for feedback on an early presentation of this work. I got the Peg Solitaire example via Tony Hoare and Bill Roscoe of Oxford University. This research is supported by EPSRC Project GR/R27105/01 entitled *Fully Expansive Proof and Algorithmic Verification*.

[7] Some examples were much worse than this and some better.

References

1. Mark D. Aagaard, Robert B. Jones, and Carl-Johan H. Seger. Lifted-FL: A Pragmatic Implementation of Combined Model Checking and Theorem Proving. In *Theorem Proving in Higher Order Logics (TPHOLs99)*, number 1690 in Lecture Notes in Computer Science, pages 323–340. Springer-Verlag, 1999.
2. Bruno Barras. Programming and computing in HOL. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 17–37. Springer-Verlag, 2000.
3. See web page `http://www-cad.eecs.berkeley.edu/∼kenmcmil/smv /`.
4. L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham. The prosper toolkit. In S. Graf and M. Schwartbach, editors, *Tools and Algorithms for Constructing Systems (TACAS 2000)*, number 1785 in Lecture Notes in Computer Science, pages 78–92. Springer-Verlag, 2000.
5. Michael J. C. Gordon. Reachability programming in HOL using BDDs. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 180–197. Springer-Verlag, 2000.
6. Mike Gordon. Reachability programming in HOL98 using BDDs. In *The 13th International Conference on Theorem Proving and Higher Order Logics*. Springer-Verlag, 2000.
7. John Harrison. Binary decision diagrams as a HOL derived rule. *The Computer Journal*, 38:162–170, 1995.
8. J. Joyce and C. Seger. The HOL-Voss System: Model-Checking inside a General-Purpose Theorem-Prover. In J. J. Joyce and C.-J. H. Seger, editors, *Higher Order Logic Theorem Proving and its Applications: 6th International Workshop, HUG'93, Vancouver, B.C., August 11-13 1993*, volume 780 of *Lecture Notes in Computer Science*, pages 185–198. Spinger-Verlag, 1994.
9. Moscow ML interface to BuDDy by Ken Friis Larsen and Jakob Lichtenberg documented at `http://www.it-c.dk/research/muddy/`.
10. K.L. McMillan. A compositional rule for hardware design refinement. In Orna Grumberg, editor, *Computer-Aided Verification, CAV '97*, Lecture Notes in Computer Science, pages 24–35, Haifa, Israel, June 1997. Springer-Verlag.
11. John O'Leary, Xudong Zhao, Robert Gerth, and Carl-Johan H. Seger. Formally verifying IEEE compliance of floating-point hardware. *Intel Technology Journal*, First Quarter 1999. Online at `http://developer.intel.com/technology/itj/`.
12. See web page `http://www.csl.sri.com/pvs.html`.
13. S. Rajan, N. Shankar, and M.K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Liege, Belgium, June 1995. Springer-Verlag.
14. Carl-Johan H. Seger. Voss - a formal hardware verification system: User's guide. Technical Report UBC TR 93-45, The University of British Columbia, December 1993.
15. Fabio Somenzi's CUDD: CU Decision Diagram Package documented at `http://vlsi.colorado.edu/ fabio/CUDD/`.

Appendix1: analysis and a solution to HexSolitaire found by `PuzzleTool`

`PuzzleTool` finds the following set of reachable end states for HexSolitaire:

```
|- Reachable HexSolitaireTrans HexSolitaireInit s /\ End HexSolitaireTrans s =
   s SUBSET HexSolitaireBoard /\
   s IN
   {{(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (2,4); (0,2)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (4,4); (0,2); (2,2); (4,2)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (3,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (3,3); (0,2); (1,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (0,2); (4,2); (2,0)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (0,4); (0,2); (3,1)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (2,4); (4,4); (4,2)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (2,4); (0,2); (3,1)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (2,4); (4,2); (1,1)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (2,4); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (4,4); (1,3); (4,2); (3,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (4,4); (0,2); (4,2); (2,0)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (4,4); (4,2); (1,1)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (1,3); (4,2); (2,0)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (1,3); (3,1)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (3,3); (0,2); (2,0)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (3,3); (1,1)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,2); (2,2); (4,2)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (0,2)};
    {(2,8); (1,7); (3,7); (0,6); (4,6); (2,2)}; {(2,8); (1,7); (3,7); (0,6); (4,6); (4,2)};
    {(2,8); (1,7); (0,6); (4,6); (3,5); (2,4); (1,3); (0,2); (3,1)}; {(2,8); (1,7); (0,6); (4,6); (0,4); (2,4); (0,2); (3,1)};
    {(2,8); (1,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (0,4); (3,3); (0,2); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (0,4); (0,2); (2,2); (4,2)}; {(2,8); (1,7); (0,6); (4,6); (0,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (0,4); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (0,4); (0,2)};
    {(2,8); (1,7); (0,6); (4,6); (2,4); (4,4); (4,2); (1,1)}; {(2,8); (1,7); (0,6); (4,6); (2,4); (0,2)};
    {(2,8); (1,7); (0,6); (4,6); (2,4); (4,2)}; {(2,8); (1,7); (0,6); (4,6); (4,4); (1,3); (4,2); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (4,4); (0,2); (2,2); (4,2)}; {(2,8); (1,7); (0,6); (4,6); (4,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (4,4); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (4,4); (4,2)};
    {(2,8); (1,7); (0,6); (4,6); (1,3); (3,3); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (1,3); (4,2); (3,1); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (1,3)}; {(2,8); (1,7); (0,6); (4,6); (3,3); (0,2); (1,1); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (3,3)};
    {(2,8); (1,7); (0,6); (4,6); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (4,6); (0,2); (4,2); (2,0)};
    {(2,8); (1,7); (0,6); (4,6); (0,2); (3,1)}; {(2,8); (1,7); (0,6); (4,6); (4,2); (1,1)}; {(2,8); (1,7); (0,6); (4,6); (2,0)};
    {(2,8); (1,7); (0,6); (3,5); (0,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (3,5); (0,4); (0,2); (4,2); (2,0)};
    {(2,8); (1,7); (0,6); (3,5); (0,4); (0,2); (3,1)}; {(2,8); (1,7); (0,6); (3,5); (1,3); (4,2); (2,0)};
    {(2,8); (1,7); (0,6); (3,5); (0,2); (2,2); (4,2)}; {(2,8); (1,7); (0,6); (3,5); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (1,7); (0,6); (3,5); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (3,5); (0,2)}; {(2,8); (1,7); (0,6); (3,5); (2,2)};
    {(2,8); (1,7); (0,6); (3,5); (4,2)}; {(2,8); (1,7); (0,6); (0,4); (2,4); (4,4); (0,2); (2,0)};
    {(2,8); (1,7); (0,6); (0,4); (2,4); (0,2); (4,2)}; {(2,8); (1,7); (0,6); (0,4); (2,4); (0,2); (1,1); (2,0)};
    {(2,8); (1,7); (0,6); (0,4); (4,4); (0,2); (2,2)}; {(2,8); (1,7); (0,6); (0,4); (4,4); (0,2); (1,1); (2,0)};
    {(2,8); (1,7); (0,6); (0,4); (3,3); (0,2)}; {(2,8); (1,7); (0,6); (0,4); (0,2); (2,0)};
    {(2,8); (1,7); (0,6); (2,4); (4,4); (0,2); (1,1); (2,0)}; {(2,8); (1,7); (0,6); (2,4); (4,4); (1,1); (3,1)};
    {(2,8); (1,7); (0,6); (2,4); (4,4)}; {(2,8); (1,7); (0,6); (2,4); (0,2); (2,0)}; {(2,8); (1,7); (0,6); (2,4); (4,2); (2,0)};
    {(2,8); (1,7); (0,6); (2,4); (1,1)}; {(2,8); (1,7); (0,6); (2,4); (3,1)}; {(2,8); (1,7); (0,6); (4,4); (0,2); (2,0)};
    {(2,8); (1,7); (0,6); (4,4); (1,1)}; {(2,8); (1,7); (0,6); (4,4); (3,1)}; {(2,8); (1,7); (0,6); (1,3); (3,3)};
    {(2,8); (1,7); (0,6); (1,3); (2,0)}; {(2,8); (1,7); (0,6); (3,3); (2,0)}; {(2,8); (1,7); (0,6); (0,2); (2,2)};
    {(2,8); (1,7); (0,6); (0,2); (4,2)}; {(2,8); (1,7); (0,6); (0,2); (1,1); (2,0)}; {(2,8); (1,7); (0,6); (2,2); (4,2)};
    {(2,8); (1,7); (0,6); (4,2); (3,1); (2,0)}; {(2,8); (1,7); (0,6); (1,1); (3,1)}; {(2,8); (1,7); (0,6)};
    {(2,8); (3,7); (0,6); (4,6); (1,5); (2,4); (3,3); (4,2); (1,1)}; {(2,8); (3,7); (0,6); (4,6); (0,4); (2,4); (0,2); (3,1)};
    {(2,8); (3,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (0,4); (3,3); (0,2); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (0,4); (0,2); (2,2); (4,2)}; {(2,8); (3,7); (0,6); (4,6); (0,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (0,4); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (0,4); (0,2)};
    {(2,8); (3,7); (0,6); (4,6); (2,4); (4,4); (4,2); (1,1)}; {(2,8); (3,7); (0,6); (4,6); (2,4); (0,2)};
    {(2,8); (3,7); (0,6); (4,6); (2,4); (4,2)}; {(2,8); (3,7); (0,6); (4,6); (4,4); (1,3); (4,2); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (4,4); (0,2); (2,2); (4,2)}; {(2,8); (3,7); (0,6); (4,6); (4,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (4,4); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (4,4); (4,2)};
    {(2,8); (3,7); (0,6); (4,6); (1,3); (3,3); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (1,3); (4,2); (3,1); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (1,3)}; {(2,8); (3,7); (0,6); (4,6); (3,3); (0,2); (1,1); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (3,3)};
    {(2,8); (3,7); (0,6); (4,6); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (3,7); (0,6); (4,6); (0,2); (4,2); (2,0)};
    {(2,8); (3,7); (0,6); (4,6); (0,2); (3,1)}; {(2,8); (3,7); (0,6); (4,6); (4,2); (1,1)}; {(2,8); (3,7); (0,6); (4,6); (2,0)};
    {(2,8); (3,7); (4,6); (1,5); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (1,5); (4,4); (0,2); (4,2); (2,0)};
    {(2,8); (3,7); (4,6); (1,5); (4,4); (4,2); (1,1)}; {(2,8); (3,7); (4,6); (1,5); (3,3); (0,2); (2,0)};
    {(2,8); (3,7); (4,6); (1,5); (0,2); (2,2); (4,2)}; {(2,8); (3,7); (4,6); (1,5); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (3,7); (4,6); (1,5); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (1,5); (0,2)}; {(2,8); (3,7); (4,6); (1,5); (2,2)};
    {(2,8); (3,7); (4,6); (1,5); (4,2)}; {(2,8); (3,7); (4,6); (0,4); (2,4); (4,4); (4,2); (2,0)};
    {(2,8); (3,7); (4,6); (0,4); (2,4); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (0,4); (2,4); (1,1); (3,1)};
    {(2,8); (3,7); (4,6); (0,4); (2,4)}; {(2,8); (3,7); (4,6); (0,4); (4,4); (2,2); (4,2)};
    {(2,8); (3,7); (4,6); (0,4); (4,4); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (0,4); (4,2); (2,0)};
    {(2,8); (3,7); (4,6); (0,4); (1,1)}; {(2,8); (3,7); (4,6); (0,4); (3,1)}; {(2,8); (3,7); (4,6); (2,4); (4,4); (0,2); (4,2)};
    {(2,8); (3,7); (4,6); (2,4); (4,4); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (2,4); (0,2); (2,0)};
    {(2,8); (3,7); (4,6); (2,4); (4,2); (2,0)}; {(2,8); (3,7); (4,6); (2,4); (1,1)}; {(2,8); (3,7); (4,6); (2,4); (3,1)};
    {(2,8); (3,7); (4,6); (4,4); (1,3); (4,2)}; {(2,8); (3,7); (4,6); (4,4); (4,2); (2,0)}; {(2,8); (3,7); (4,6); (1,3); (3,3)};
    {(2,8); (3,7); (4,6); (1,3); (2,0)}; {(2,8); (3,7); (4,6); (3,3); (2,0)}; {(2,8); (3,7); (4,6); (0,2); (2,2)};
    {(2,8); (3,7); (4,6); (0,2); (4,2)}; {(2,8); (3,7); (4,6); (0,2); (1,1); (2,0)}; {(2,8); (3,7); (4,6); (2,2); (4,2)};
    {(2,8); (3,7); (4,6); (4,2); (3,1); (2,0)}; {(2,8); (3,7); (4,6); (1,1); (3,1)}; {(2,8); (3,7); (4,6)};
    {(2,8); (0,6); (2,6); (0,4); (2,4); (4,4); (0,2); (2,2); (2,0)}; {(2,8); (0,6); (2,6); (2,4); (0,2); (2,2); (2,0)};
    {(2,8); (0,6); (2,6); (2,4); (2,2); (4,2); (2,0)}; {(2,8); (0,6); (4,6); (1,5); (2,4); (3,3); (4,2)};
    {(2,8); (0,6); (4,6); (3,5); (2,4); (1,3); (0,2)}; {(2,8); (0,6); (4,6); (0,4); (2,4); (0,2)};
    {(2,8); (0,6); (4,6); (0,4); (4,4); (0,2); (2,2); (4,2)}; {(2,8); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (0,6); (4,6); (0,4); (4,4); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (0,6); (4,6); (0,4); (3,3); (0,2); (1,1); (2,0)};
    {(2,8); (0,6); (4,6); (0,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (0,6); (4,6); (0,4); (0,2); (4,2); (2,0)};
    {(2,8); (0,6); (4,6); (0,4); (0,2); (3,1)}; {(2,8); (0,6); (4,6); (2,4); (4,4); (4,2)}; {(2,8); (0,6); (4,6); (2,4); (0,2); (3,1)};
    {(2,8); (0,6); (4,6); (2,4); (4,2); (1,1)}; {(2,8); (0,6); (4,6); (2,4); (2,0)};
    {(2,8); (0,6); (4,6); (4,4); (1,3); (4,2); (3,1); (2,0)}; {(2,8); (0,6); (4,6); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
    {(2,8); (0,6); (4,6); (4,4); (0,2); (4,2); (2,0)}; {(2,8); (0,6); (4,6); (4,4); (4,2); (1,1)};
    {(2,8); (0,6); (4,6); (1,3); (4,2); (2,0)}; {(2,8); (0,6); (4,6); (1,3); (3,1)}; {(2,8); (0,6); (4,6); (3,3); (0,2); (2,0)};
    {(2,8); (0,6); (4,6); (3,3); (1,1)}; {(2,8); (0,6); (4,6); (0,2); (2,2); (4,2)}; {(2,8); (0,6); (4,6); (0,2); (4,2); (1,1); (2,0)};
    {(2,8); (0,6); (4,6); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (0,6); (4,6); (0,2)}; {(2,8); (0,6); (4,6); (2,2)};
    {(2,8); (0,6); (4,6); (4,2)}; {(2,8); (0,6); (1,5); (2,4); (3,3); (4,2); (2,0)}; {(2,8); (0,6); (3,5); (0,4); (0,2); (2,2); (4,2)};
    {(2,8); (0,6); (3,5); (0,4); (0,2); (4,2); (1,1); (2,0)}; {(2,8); (0,6); (3,5); (0,4); (0,2); (4,2); (3,1); (2,0)};
    {(2,8); (0,6); (3,5); (0,4); (0,2)}; {(2,8); (0,6); (3,5); (1,3); (4,2); (3,1); (2,0)}; {(2,8); (0,6); (3,5); (1,3)};
    {(2,8); (0,6); (3,5); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (0,6); (3,5); (0,2); (4,2); (2,0)};
    {(2,8); (0,6); (3,5); (0,2); (3,1)}; {(2,8); (0,6); (3,5); (4,2); (1,1)}; {(2,8); (0,6); (3,5); (2,0)};
    {(2,8); (0,6); (0,4); (2,4); (4,4); (0,2); (1,1); (2,0)}; {(2,8); (0,6); (0,4); (2,4); (0,2); (2,0)};
    {(2,8); (0,6); (0,4); (4,4); (0,2); (2,0)}; {(2,8); (0,6); (0,4); (0,2); (2,2)}; {(2,8); (0,6); (0,4); (0,2); (4,2)};
```

{(2,8); (0,6); (0,4); (0,2); (1,1); (2,0)}; {(2,8); (0,6); (2,4); (4,4); (0,2); (2,0)}; {(2,8); (0,6); (2,4); (4,4); (1,1)};
{(2,8); (0,6); (2,4); (4,4); (3,1)}; {(2,8); (0,6); (2,4); (0,2); (4,2)}; {(2,8); (0,6); (2,4); (0,2); (1,1); (2,0)};
{(2,8); (0,6); (2,4); (4,2); (3,1); (2,0)}; {(2,8); (0,6); (2,4); (1,1); (3,1)}; {(2,8); (0,6); (2,4)};
{(2,8); (0,6); (4,4); (1,3); (2,0)}; {(2,8); (0,6); (4,4); (3,3); (2,2); (1,1)}; {(2,8); (0,6); (4,4); (0,2); (2,2)};
{(2,8); (0,6); (4,4); (0,2); (1,1); (2,0)}; {(2,8); (0,6); (4,4); (1,1); (3,1)}; {(2,8); (0,6); (4,4)}; {(2,8); (0,6); (1,3); (4,2)};
{(2,8); (0,6); (3,3); (0,2)}; {(2,8); (0,6); (0,2); (2,0)}; {(2,8); (0,6); (4,2); (2,0)}; {(2,8); (0,6); (1,1)}; {(2,8); (0,6); (3,1)};
{(2,8); (2,6); (4,6); (0,4); (2,4); (4,4); (2,2); (4,2); (2,0)}; {(2,8); (2,6); (4,6); (2,4); (0,2); (2,2); (2,0)};
{(2,8); (2,6); (4,6); (2,4); (2,2); (4,2); (2,0)}; {(2,8); (2,6); (0,4); (2,4); (2,2); (2,0)};
{(2,8); (2,6); (2,4); (4,4); (2,2); (2,0)}; {(2,8); (4,6); (1,5); (4,4); (0,2); (2,2); (4,2)};
{(2,8); (4,6); (1,5); (4,4); (0,2); (4,2); (1,1); (2,0)}; {(2,8); (4,6); (1,5); (4,4); (0,2); (4,2); (3,1); (2,0)};
{(2,8); (4,6); (1,5); (4,4); (4,2)}; {(2,8); (4,6); (1,5); (3,3); (0,2); (1,1); (2,0)}; {(2,8); (4,6); (1,5); (3,3)};
{(2,8); (4,6); (1,5); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (4,6); (1,5); (0,2); (4,2); (2,0)};
{(2,8); (4,6); (1,5); (0,2); (3,1)}; {(2,8); (4,6); (1,5); (4,2); (1,1)}; {(2,8); (4,6); (1,5); (2,0)};
{(2,8); (4,6); (3,5); (2,4); (1,3); (0,2); (2,0)}; {(2,8); (4,6); (0,4); (2,4); (4,4); (4,2); (3,1); (2,0)};
{(2,8); (4,6); (0,4); (2,4); (4,2); (2,0)}; {(2,8); (4,6); (0,4); (2,4); (1,1)}; {(2,8); (4,6); (0,4); (2,4); (3,1)};
{(2,8); (4,6); (0,4); (4,4); (4,2); (2,0)}; {(2,8); (4,6); (0,4); (1,3); (2,2); (3,1)}; {(2,8); (4,6); (0,4); (3,3); (2,0)};
{(2,8); (4,6); (0,4); (2,2); (4,2)}; {(2,8); (4,6); (0,4); (4,2); (3,1); (2,0)}; {(2,8); (4,6); (0,4); (1,1); (3,1)};
{(2,8); (4,6); (0,4)}; {(2,8); (4,6); (2,4); (4,4); (4,2); (2,0)}; {(2,8); (4,6); (2,4); (0,2); (4,2)};
{(2,8); (4,6); (2,4); (0,2); (1,1); (2,0)}; {(2,8); (4,6); (2,4); (4,2); (3,1); (2,0)}; {(2,8); (4,6); (2,4); (1,1); (3,1)};
{(2,8); (4,6); (2,4)}; {(2,8); (4,6); (4,4); (0,2); (4,2)}; {(2,8); (4,6); (4,4); (2,2); (4,2)};
{(2,8); (4,6); (4,4); (4,2); (3,1); (2,0)}; {(2,8); (4,6); (1,3); (4,2)}; {(2,8); (4,6); (3,3); (0,2)}; {(2,8); (4,6); (0,2); (2,0)};
{(2,8); (4,6); (4,2); (2,0)}; {(2,8); (4,6); (1,1)}; {(2,8); (4,6); (3,1)}; {(2,8); (1,5); (3,5); (0,2); (2,2); (4,2)};
{(2,8); (1,5); (3,5); (0,2); (4,2); (1,1); (2,0)}; {(2,8); (1,5); (3,5); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (1,5); (3,5); (0,2)};
{(2,8); (1,5); (3,5); (4,2)}; {(2,8); (1,5); (4,4); (0,2); (2,0)}; {(2,8); (1,5); (4,4); (1,1)}; {(2,8); (1,5); (4,4); (3,1)};
{(2,8); (1,5); (3,3); (2,0)}; {(2,8); (1,5); (0,2); (2,2)}; {(2,8); (1,5); (0,2); (4,2)}; {(2,8); (1,5); (0,2); (1,1); (2,0)};
{(2,8); (1,5); (2,2); (4,2)}; {(2,8); (1,5); (4,2); (3,1); (2,0)}; {(2,8); (1,5); (1,1); (3,1)}; {(2,8); (1,5)};
{(2,8); (3,5); (0,4); (4,2); (2,0)}; {(2,8); (3,5); (0,4); (1,1)}; {(2,8); (3,5); (0,4); (3,1)}; {(2,8); (3,5); (1,3); (2,0)};
{(2,8); (3,5); (0,2); (2,2)}; {(2,8); (3,5); (0,2); (4,2)}; {(2,8); (3,5); (0,2); (1,1); (2,0)}; {(2,8); (3,5); (2,2); (4,2)};
{(2,8); (3,5); (4,2); (3,1); (2,0)}; {(2,8); (3,5); (1,1); (3,1)}; {(2,8); (3,5)}; {(2,8); (0,4); (2,4); (4,4); (2,0)};
{(2,8); (0,4); (2,4); (4,2)}; {(2,8); (0,4); (4,4); (2,2)}; {(2,8); (0,4); (3,3)}; {(2,8); (0,4); (4,2); (1,1)}; {(2,8); (0,4); (2,0)};
{(2,8); (2,4); (4,4); (0,2)}; {(2,8); (2,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(2,8); (2,4); (0,2); (4,2); (2,0)};
{(2,8); (2,4); (0,2); (3,1)}; {(2,8); (2,4); (4,2); (1,1)}; {(2,8); (2,4); (2,0)}; {(2,8); (4,4); (1,3)}; {(2,8); (4,4); (0,2); (3,1)};
{(2,8); (4,4); (2,0)}; {(2,8); (1,3); (4,2); (2,0)}; {(2,8); (1,3); (3,1)}; {(2,8); (3,3); (0,2); (2,0)}; {(2,8); (3,3); (1,1)};
{(2,8); (0,2); (2,2); (4,2)}; {(2,8); (0,2); (4,2); (1,1); (2,0)}; {(2,8); (0,2); (4,2); (3,1); (2,0)}; {(2,8); (0,2)}; {(2,8); (2,2)};
{(2,8); (4,2)}; {(1,7); (3,7); (0,4); (2,4); (4,2); (3,1); (2,0)}; {(1,7); (3,7); (0,4); (2,4)}; {(1,7); (3,7); (0,4); (4,2); (2,0)};
{(1,7); (3,7); (2,4); (4,4); (0,2); (1,1); (2,0)}; {(1,7); (3,7); (2,4); (4,4)}; {(1,7); (3,7); (2,4); (0,2); (2,0)};
{(1,7); (3,7); (2,4); (4,2); (2,0)}; {(1,7); (3,7); (2,4); (1,1)}; {(1,7); (3,7); (2,4); (3,1)}; {(1,7); (3,7); (4,4); (0,2); (2,0)};
{(1,7); (3,7); (1,3); (2,0)}; {(1,7); (3,7); (3,3); (2,0)}; {(1,7); (3,7); (0,2); (2,2)}; {(1,7); (3,7); (0,2); (4,2)};
{(1,7); (3,7); (0,2); (1,1); (2,0)}; {(1,7); (3,7); (2,2); (4,2)}; {(1,7); (3,7); (4,2); (3,1); (2,0)};
{(1,7); (2,6); (3,5); (4,4); (0,2); (2,0)}; {(1,7); (4,6); (1,5); (1,3); (4,2); (1,1)};
{(1,7); (4,6); (3,5); (2,4); (1,3); (0,2); (4,2); (3,1); (2,0)}; {(1,7); (4,6); (3,5); (2,4); (1,3); (0,2)};
{(1,7); (4,6); (0,4); (2,4); (4,4); (4,2); (1,1)}; {(1,7); (4,6); (0,4); (2,4); (4,2)}; {(1,7); (4,6); (0,4); (4,4); (4,2)};
{(1,7); (4,6); (0,4); (3,3)}; {(1,7); (4,6); (0,4); (4,2); (1,1)}; {(1,7); (4,6); (0,4); (2,0)};
{(1,7); (4,6); (2,4); (4,4); (0,2); (4,2); (1,1); (2,0)}; {(1,7); (4,6); (2,4); (4,4); (0,2); (4,2); (3,1); (2,0)};
{(1,7); (4,6); (2,4); (4,4); (4,2)}; {(1,7); (4,6); (2,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
{(1,7); (4,6); (2,4); (0,2); (4,2); (2,0)}; {(1,7); (4,6); (2,4); (0,2); (3,1)}; {(1,7); (4,6); (2,4); (4,2); (1,1)};
{(1,7); (4,6); (2,4); (2,0)}; {(1,7); (4,6); (4,4); (1,3); (4,2); (3,1); (2,0)};
{(1,7); (4,6); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(1,7); (4,6); (4,4); (0,2); (4,2); (2,0)};
{(1,7); (4,6); (4,4); (4,2); (1,1)}; {(1,7); (4,6); (1,3); (4,2); (2,0)}; {(1,7); (4,6); (1,3); (3,1)};
{(1,7); (4,6); (3,3); (0,2); (2,0)}; {(1,7); (4,6); (3,3); (1,1)}; {(1,7); (4,6); (0,2); (2,2); (4,2)};
{(1,7); (4,6); (0,2); (4,2); (1,1); (2,0)}; {(1,7); (4,6); (0,2); (4,2); (3,1); (2,0)}; {(1,7); (4,6); (0,2)}; {(1,7); (4,6); (2,2)};
{(1,7); (4,6); (4,2)}; {(1,7); (3,5); (0,4); (4,2)}; {(1,7); (3,5); (0,2); (4,2); (1,1); (3,1); (2,0)};
{(1,7); (3,5); (0,2); (4,2); (2,0)}; {(1,7); (3,5); (0,2); (3,1)}; {(1,7); (3,5); (4,2); (1,1)}; {(1,7); (3,5); (2,0)};
{(1,7); (0,4); (2,4); (4,4)}; {(1,7); (0,4); (2,4); (4,2); (2,0)}; {(1,7); (0,4); (2,4); (1,1)}; {(1,7); (0,4); (2,4); (3,1)};
{(1,7); (0,4); (3,3); (2,0)}; {(1,7); (0,4); (2,2); (4,2)}; {(1,7); (0,4); (4,2); (3,1); (2,0)}; {(1,7); (2,4); (4,4); (0,2); (2,0)};
{(1,7); (2,4); (4,4); (1,1)}; {(1,7); (2,4); (4,4); (3,1)}; {(1,7); (2,4); (0,2); (4,2)}; {(1,7); (2,4); (0,2); (1,1); (2,0)};
{(1,7); (2,4); (4,2); (3,1); (2,0)}; {(1,7); (2,4); (1,1); (3,1)}; {(1,7); (2,4)}; {(1,7); (4,4); (1,3); (2,0)};
{(1,7); (4,4); (0,2); (2,2)}; {(1,7); (4,4); (0,2); (1,1); (2,0)}; {(1,7); (1,3); (4,2)}; {(1,7); (3,3); (0,2)}; {(1,7); (0,2); (2,0)};
{(1,7); (4,2); (2,0)}; {(3,7); (0,6); (1,5); (2,4); (3,3); (0,2); (4,2); (1,1); (2,0)}; {(3,7); (0,6); (1,5); (2,4); (3,3); (4,2)};
{(3,7); (0,6); (3,5); (3,3); (0,2); (3,1)}; {(3,7); (0,6); (0,4); (2,4); (4,4); (0,2); (3,1)};
{(3,7); (0,6); (0,4); (2,4); (0,2); (4,2); (1,1); (2,0)}; {(3,7); (0,6); (0,4); (2,4); (0,2); (4,2); (3,1); (2,0)};
{(3,7); (0,6); (0,4); (2,4); (0,2)}; {(3,7); (0,6); (0,4); (4,4); (0,2)}; {(3,7); (0,6); (0,4); (3,3); (0,2); (1,1); (2,0)};
{(3,7); (0,6); (0,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(3,7); (0,6); (0,4); (0,2); (4,2); (2,0)};
{(3,7); (0,6); (0,4); (0,2); (3,1)}; {(3,7); (0,6); (2,4); (4,4); (0,2)}; {(3,7); (0,6); (2,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
{(3,7); (0,6); (2,4); (0,2); (4,2); (2,0)}; {(3,7); (0,6); (2,4); (0,2); (3,1)}; {(3,7); (0,6); (2,4); (4,2); (1,1)};
{(3,7); (0,6); (2,4); (2,0)}; {(3,7); (0,6); (4,4); (1,3)}; {(3,7); (0,6); (4,4); (0,2); (3,1)}; {(3,7); (0,6); (4,4); (2,0)};
{(3,7); (0,6); (1,3); (4,2); (2,0)}; {(3,7); (0,6); (1,3); (3,1)}; {(3,7); (0,6); (3,3); (0,2); (2,0)}; {(3,7); (0,6); (3,3); (1,1)};
{(3,7); (0,6); (0,2); (2,2); (4,2)}; {(3,7); (0,6); (0,2); (4,2); (1,1); (2,0)}; {(3,7); (0,6); (0,2); (4,2); (3,1); (2,0)};
{(3,7); (0,6); (0,2)}; {(3,7); (0,6); (2,2)}; {(3,7); (0,6); (4,2)}; {(3,7); (2,6); (1,5); (0,4); (4,2); (2,0)};
{(3,7); (1,5); (4,4); (0,2)}; {(3,7); (1,5); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(3,7); (1,5); (0,2); (4,2); (2,0)};
{(3,7); (1,5); (0,2); (3,1)}; {(3,7); (1,5); (4,2); (1,1)}; {(3,7); (1,5); (2,0)}; {(3,7); (0,4); (2,4); (4,4)};
{(3,7); (0,4); (2,4); (4,2); (2,0)}; {(3,7); (0,4); (2,4); (1,1)}; {(3,7); (0,4); (2,4); (3,1)}; {(3,7); (0,4); (3,3); (2,0)};
{(3,7); (0,4); (2,2); (4,2)}; {(3,7); (0,4); (4,2); (3,1); (2,0)}; {(3,7); (2,4); (4,4); (0,2); (2,0)}; {(3,7); (2,4); (4,4); (1,1)};
{(3,7); (2,4); (4,4); (3,1)}; {(3,7); (2,4); (0,2); (4,2)}; {(3,7); (2,4); (0,2); (1,1); (2,0)}; {(3,7); (2,4); (4,2); (3,1); (2,0)};
{(3,7); (2,4); (1,1); (3,1)}; {(3,7); (2,4)}; {(3,7); (4,4); (1,3); (2,0)}; {(3,7); (4,4); (0,2); (2,2)};
{(3,7); (4,4); (0,2); (1,1); (2,0)}; {(3,7); (1,3); (4,2)}; {(3,7); (3,3); (0,2)}; {(3,7); (0,2); (2,0)}; {(3,7); (4,2); (2,0)};
{(0,6); (2,6); (4,6); (0,4); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(0,6); (2,6); (4,6); (0,4); (4,4); (0,2); (4,2); (2,0)};
{(0,6); (2,6); (4,6); (0,4); (3,3); (0,2); (2,0)}; {(0,6); (2,6); (4,6); (0,4); (0,2); (2,2); (4,2)};
{(0,6); (2,6); (4,6); (0,4); (0,2); (4,2); (1,1); (2,0)}; {(0,6); (2,6); (4,6); (0,4); (0,2); (4,2); (3,1); (2,0)};
{(0,6); (2,6); (4,6); (0,4); (0,2)}; {(0,6); (2,6); (4,6); (4,4); (1,3); (4,2); (2,0)};
{(0,6); (2,6); (4,6); (4,4); (0,2); (2,2); (4,2)}; {(0,6); (2,6); (4,6); (4,4); (0,2); (4,2); (1,1); (2,0)};
{(0,6); (2,6); (4,6); (4,4); (0,2); (4,2); (3,1); (2,0)}; {(0,6); (2,6); (4,6); (4,4); (4,2)};
{(0,6); (2,6); (4,6); (1,3); (3,3); (2,0)}; {(0,6); (2,6); (4,6); (1,3); (4,2); (3,1); (2,0)}; {(0,6); (2,6); (4,6); (1,3)};
{(0,6); (2,6); (4,6); (3,3); (0,2); (1,1); (2,0)}; {(0,6); (2,6); (4,6); (3,3)};
{(0,6); (2,6); (4,6); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(0,6); (2,6); (4,6); (0,2); (4,2); (2,0)};
{(0,6); (2,6); (4,6); (0,2); (3,1)}; {(0,6); (2,6); (4,6); (4,2); (1,1)}; {(0,6); (2,6); (4,6); (2,0)};
{(0,6); (2,6); (0,4); (4,4); (0,2); (1,1); (2,0)}; {(0,6); (2,6); (0,4); (3,3); (0,2)}; {(0,6); (2,6); (0,4); (0,2); (2,0)};
{(0,6); (2,6); (4,4); (0,2); (2,0)}; {(0,6); (2,6); (4,4); (1,1)}; {(0,6); (2,6); (4,4); (3,1)}; {(0,6); (2,6); (1,3); (2,0)};
{(0,6); (2,6); (3,3); (2,0)}; {(0,6); (2,6); (0,2); (2,2)}; {(0,6); (2,6); (0,2); (4,2)}; {(0,6); (2,6); (0,2); (1,1); (2,0)};
{(0,6); (2,6); (2,2); (4,2)}; {(0,6); (2,6); (4,2); (3,1); (2,0)}; {(0,6); (2,6); (1,1); (3,1)}; {(0,6); (2,6)};
{(0,6); (4,6); (1,5); (2,4); (3,3); (0,2); (4,2)}; {(0,6); (4,6); (3,5); (2,4); (1,3); (0,2); (4,2)};
{(0,6); (4,6); (0,4); (2,4); (0,2); (4,2)}; {(0,6); (4,6); (0,4); (2,4); (0,2); (1,1); (2,0)};
{(0,6); (4,6); (0,4); (4,4); (0,2); (4,2)}; {(0,6); (4,6); (0,4); (3,3); (0,2)}; {(0,6); (4,6); (0,4); (0,2); (2,0)};
{(0,6); (4,6); (2,4); (4,4); (0,2); (4,2)}; {(0,6); (4,6); (2,4); (4,4); (4,2); (3,1); (2,0)}; {(0,6); (4,6); (2,4); (0,2); (2,0)};
{(0,6); (4,6); (2,4); (4,2); (2,0)}; {(0,6); (4,6); (2,4); (1,1)}; {(0,6); (4,6); (2,4); (3,1)}; {(0,6); (4,6); (4,4); (1,3); (4,2)};
{(0,6); (4,6); (4,4); (4,2); (2,0)}; {(0,6); (4,6); (1,3); (3,3)}; {(0,6); (4,6); (1,3); (2,0)}; {(0,6); (4,6); (3,3); (2,0)};
{(0,6); (4,6); (0,2); (2,2)}; {(0,6); (4,6); (0,2); (4,2)}; {(0,6); (4,6); (0,2); (1,1); (2,0)}; {(0,6); (4,6); (2,2); (4,2)};
{(0,6); (4,6); (4,2); (3,1); (2,0)}; {(0,6); (4,6); (1,1); (3,1)}; {(0,6); (4,6)}; {(0,6); (1,5); (2,4); (3,3); (0,2); (4,2); (2,0)};
{(0,6); (1,5); (2,4); (3,3); (4,2); (1,1)}; {(0,6); (3,5); (0,4); (0,2); (2,2)}; {(0,6); (3,5); (0,4); (0,2); (4,2)};
{(0,6); (3,5); (0,4); (0,2); (1,1); (2,0)}; {(0,6); (3,5); (1,3); (4,2)}; {(0,6); (3,5); (0,2); (2,0)}; {(0,6); (3,5); (4,2); (2,0)};
{(0,6); (3,5); (1,1)}; {(0,6); (3,5); (3,1)}; {(0,6); (0,4); (2,4); (4,4); (0,2)};
{(0,6); (0,4); (2,4); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(0,6); (0,4); (2,4); (0,2); (4,2); (2,0)};
{(0,6); (0,4); (2,4); (0,2); (3,1)}; {(0,6); (0,4); (4,4); (0,2); (3,1)}; {(0,6); (0,4); (3,3); (0,2); (2,0)};
{(0,6); (0,4); (0,2); (2,2); (4,2)}; {(0,6); (0,4); (0,2); (4,2); (1,1); (2,0)}; {(0,6); (0,4); (0,2); (4,2); (3,1); (2,0)};
{(0,6); (0,4); (0,2)}; {(0,6); (2,4); (4,4); (0,2); (3,1)}; {(0,6); (2,4); (4,4); (2,0)}; {(0,6); (2,4); (0,2); (4,2); (1,1); (2,0)};

```
{(0,6); (2,4); (0,2); (4,2); (3,1); (2,0)}; {(0,6); (2,4); (0,2)}; {(0,6); (2,4); (4,2)}; {(0,6); (4,4); (1,3); (3,1)};
{(0,6); (4,4); (0,2)}; {(0,6); (4,4); (2,2)}; {(0,6); (1,3); (3,3); (2,0)}; {(0,6); (1,3); (4,2); (3,1); (2,0)}; {(0,6); (1,3)};
{(0,6); (3,3); (0,2); (1,1); (2,0)}; {(0,6); (3,3)}; {(0,6); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(0,6); (0,2); (4,2); (2,0)};
{(0,6); (0,2); (3,1)}; {(0,6); (4,2); (1,1)}; {(0,6); (2,0)}; {(2,6); (4,6); (0,4); (4,4); (4,2); (3,1); (2,0)};
{(2,6); (4,6); (0,4); (4,2); (2,0)}; {(2,6); (4,6); (0,4); (1,1)}; {(2,6); (4,6); (0,4); (3,1)}; {(2,6); (4,6); (4,4); (1,3); (4,2)};
{(2,6); (4,6); (4,4); (4,2); (2,0)}; {(2,6); (4,6); (1,3); (2,0)}; {(2,6); (4,6); (3,3); (2,0)}; {(2,6); (4,6); (0,2); (2,2)};
{(2,6); (4,6); (0,2); (4,2)}; {(2,6); (4,6); (0,2); (1,1); (2,0)}; {(2,6); (4,6); (2,2); (4,2)}; {(2,6); (4,6); (4,2); (3,1); (2,0)};
{(2,6); (4,6); (1,1); (3,1)}; {(2,6); (4,6)}; {(2,6); (0,4); (4,4); (2,0)}; {(2,6); (0,4); (4,2)}; {(2,6); (4,4); (0,2)};
{(2,6); (1,3); (4,2); (3,1); (2,0)}; {(2,6); (3,3); (0,2); (1,1); (2,0)}; {(2,6); (0,2); (4,2); (1,1); (3,1); (2,0)};
{(2,6); (0,2); (4,2); (2,0)}; {(2,6); (0,2); (3,1)}; {(2,6); (4,2); (1,1)}; {(2,6); (2,0)}; {(4,6); (1,5); (4,4); (0,2); (4,2)};
{(4,6); (1,5); (4,4); (2,2); (4,2)}; {(4,6); (1,5); (4,4); (4,2); (3,1); (2,0)}; {(4,6); (1,5); (3,3); (0,2)};
{(4,6); (1,5); (0,2); (2,0)}; {(4,6); (1,5); (4,2); (2,0)}; {(4,6); (1,5); (1,1)}; {(4,6); (1,5); (3,1)};
{(4,6); (3,5); (2,4); (1,3); (0,2); (4,2); (2,0)}; {(4,6); (3,5); (2,4); (1,3); (0,2); (3,1)}; {(4,6); (0,4); (2,4); (4,4); (4,2)};
{(4,6); (0,4); (2,4); (4,2); (1,1)}; {(4,6); (0,4); (2,4); (2,0)}; {(4,6); (0,4); (4,4); (4,2); (1,1)}; {(4,6); (0,4); (3,3); (1,1)};
{(4,6); (0,4); (2,2)}; {(4,6); (0,4); (4,2)}; {(4,6); (2,4); (4,4); (0,2); (4,2); (1,1); (3,1); (2,0)};
{(4,6); (2,4); (4,4); (0,2); (4,2); (2,0)}; {(4,6); (2,4); (4,4); (4,2); (1,1)}; {(4,6); (2,4); (0,2); (4,2); (1,1); (2,0)};
{(4,6); (2,4); (0,2); (4,2); (3,1); (2,0)}; {(4,6); (2,4); (0,2)}; {(4,6); (2,4); (4,2)}; {(4,6); (4,4); (1,3); (4,2); (2,0)};
{(4,6); (4,4); (0,2); (2,2); (4,2)}; {(4,6); (4,4); (0,2); (4,2); (1,1); (2,0)}; {(4,6); (4,4); (0,2); (4,2); (3,1); (2,0)};
{(4,6); (4,4); (4,2)}; {(4,6); (1,3); (3,3); (2,0)}; {(4,6); (1,3); (4,2); (3,1); (2,0)}; {(4,6); (1,3)};
{(4,6); (3,3); (0,2); (1,1); (2,0)}; {(4,6); (3,3)}; {(4,6); (0,2); (4,2); (1,1); (3,1); (2,0)}; {(4,6); (0,2); (4,2); (2,0)};
{(4,6); (0,2); (3,1)}; {(4,6); (4,2); (1,1)}; {(4,6); (2,0)}; {(1,5); (3,5); (0,2); (4,2)}; {(1,5); (3,5); (0,2); (1,1); (2,0)};
{(1,5); (3,5); (4,2); (3,1); (2,0)}; {(1,5); (4,4); (0,2); (3,1)}; {(1,5); (4,4); (2,0)}; {(1,5); (3,3); (0,2); (2,0)};
{(1,5); (0,2); (2,2); (4,2)}; {(1,5); (0,2); (4,2); (1,1); (2,0)}; {(1,5); (0,2); (4,2); (3,1); (2,0)}; {(1,5); (0,2)}; {(1,5); (4,2)};
{(3,5); (0,4); (4,2); (1,1)}; {(3,5); (0,4); (2,0)}; {(3,5); (1,3); (4,2); (2,0)}; {(3,5); (0,2); (2,2); (4,2)};
{(3,5); (0,2); (4,2); (1,1); (2,0)}; {(3,5); (0,2); (4,2); (3,1); (2,0)}; {(3,5); (0,2)}; {(3,5); (4,2)}; {(0,4); (2,4); (4,4); (1,1)};
{(0,4); (2,4); (4,4); (3,1)}; {(0,4); (2,4); (4,2); (3,1); (2,0)}; {(0,4); (2,4); (1,1); (3,1)}; {(0,4); (2,4)}; {(0,4); (4,2); (2,0)};
{(2,4); (4,4); (0,2); (1,1); (2,0)}; {(2,4); (4,4); (1,1); (3,1)}; {(2,4); (4,4)}; {(2,4); (0,2); (2,0)}; {(2,4); (4,2); (2,0)};
{(2,4); (1,1)}; {(2,4); (3,1)}; {(4,4); (0,2); (2,0)}; {(1,3); (2,0)}; {(3,3); (2,0)}; {(0,2); (2,2)}; {(0,2); (4,2)};
{(0,2); (1,1); (2,0)}; {(2,2); (4,2)}; {(4,2); (3,1); (2,0)}} : thm
```

The reachable end states with two or fewer pegs are:

```
{{(2,8), (1,5)}, {(2,8), (3,5)}, {(2,8), (0,2)}, {(2,8), (2,2)}, {(2,8), (4,2)}, {(1,7), (2,4)},
 {(3,7), (2,4)}, {(0,6), (2,6)}, {(0,6), (4,6)}, {(0,6), (1,3)}, {(0,6), (3,3)}, {(0,6), (2,0)},
 {(2,6), (4,6)}, {(2,6), (2,0)}, {(4,6), (1,3)}, {(4,6), (3,3)}, {(4,6), (2,0)}, {(1,5), (0,2)},
 {(1,5), (4,2)}, {(3,5), (0,2)}, {(3,5), (4,2)}, {(0,4), (2,4)}, {(2,4), (4,4)}, {(2,4), (1,1)},
 {(2,4), (3,1)}, {(1,3), (2,0)}, {(3,3), (2,0)}, {(0,2), (2,2)}, {(0,2), (4,2)}, {(2,2), (4,2)}}
```

`PuzzleTool` finds the following solution to reaching `{(2,4); (4,4)}`:

```
|- !state. HexSolitaireFinalState = {(2,4); (4,4)} : thm

|- HexSolitaireInit
     {(2,8); (1,7); (3,7); (0,6); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)} /\
   HexSolitaireTrans
     ({(2,8); (1,7); (3,7); (0,6); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(2,8); (1,7); (3,7); (0,6); (2,6); (1,5); (0,4); (2,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(2,8); (1,7); (3,7); (0,6); (2,6); (1,5); (0,4); (2,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(1,7); (0,6); (2,6); (4,6); (1,5); (0,4); (2,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(1,7); (0,6); (2,6); (4,6); (1,5); (0,4); (2,4); (4,4); (1,3); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(1,7); (0,6); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(1,7); (0,6); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(1,7); (2,6); (4,6); (3,5); (0,4); (2,4); (4,4); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(1,7); (2,6); (4,6); (3,5); (0,4); (2,4); (4,4); (3,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(1,7); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(1,7); (2,6); (4,6); (1,5); (3,5); (0,4); (4,4); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(2,6); (4,6); (3,5); (0,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)}) /\
   HexSolitaireTrans
     ({(2,6); (4,6); (3,5); (0,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1); (2,0)},
      {(2,6); (4,6); (3,5); (0,4); (2,4); (4,4); (1,3); (0,2); (4,2); (1,1); (3,1)}) /\
   HexSolitaireTrans
     ({(2,6); (4,6); (3,5); (0,4); (2,4); (4,4); (1,3); (0,2); (4,2); (1,1); (3,1)},
      {(4,6); (3,5); (0,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1)}) /\
   HexSolitaireTrans
     ({(4,6); (3,5); (0,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1)},
      {(0,4); (2,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1)}) /\
   HexSolitaireTrans
     ({(0,4); (2,4); (4,4); (1,3); (0,2); (2,2); (4,2); (1,1); (3,1)},{(0,4); (2,4); (4,4); (1,3); (2,2); (4,2); (3,1); (2,0)}) /\
   HexSolitaireTrans ({(0,4); (2,4); (4,4); (1,3); (2,2); (4,2); (3,1); (2,0)},{(3,5); (0,4); (4,4); (2,2); (4,2); (3,1); (2,0)}) /\
   HexSolitaireTrans ({(3,5); (0,4); (4,4); (2,2); (4,2); (3,1); (2,0)},{(3,5); (0,4); (2,4); (4,4); (4,2); (3,1)}) /\
   HexSolitaireTrans ({(3,5); (0,4); (2,4); (4,4); (4,2); (3,1)},{(0,4); (4,4); (1,3); (4,2); (3,1)}) /\
   HexSolitaireTrans ({(0,4); (4,4); (1,3); (4,2); (3,1)},{(4,4); (2,2); (4,2); (3,1)}) /\
   HexSolitaireTrans ({(4,4); (2,2); (4,2); (3,1)},{(4,4); (2,2); (2,0)}) /\
   HexSolitaireTrans ({(4,4); (2,2); (2,0)},{(2,4); (4,4)}) /\ HexSolitaireFinal {(2,4); (4,4)} : thm
```

Appendix 2: solution to Peg Solitaire found by `PuzzleTool`

`PuzzleTool` finds the following set of reachable end states for PegSolitaire:

```
|- SolitaireInit
     {(2,6); (3,6); (4,6); (2,5); (3,5); (4,5); (0,4); (1,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (4,3); (5,3); (6,3);
      (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)} /\
   SolitaireTrans
     ({(2,6); (3,6); (4,6); (2,5); (3,5); (4,5); (0,4); (1,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (4,3); (5,3); (6,3);
       (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(2,6); (3,6); (4,6); (2,5); (4,5); (0,4); (1,4); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2);
       (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(2,6); (3,6); (4,6); (2,5); (4,5); (0,4); (1,4); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2);
       (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(2,6); (3,6); (4,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2);
       (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(2,6); (3,6); (4,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2);
       (2,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(3,6); (4,6); (4,5); (0,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2);
       (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(3,6); (4,6); (4,5); (0,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2);
       (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(3,6); (4,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2);
       (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(3,6); (4,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2);
       (4,2); (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(2,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2);
       (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(2,6); (2,5); (4,5); (0,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2);
       (5,2); (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(4,5); (0,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2);
       (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (0,4); (2,4); (3,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2);
       (6,2); (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(4,5); (0,4); (1,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2);
       (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (0,4); (1,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2);
       (2,1); (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(4,5); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1);
       (3,1); (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (2,1);
       (3,1); (4,1); (2,0); (3,0); (4,0)},
      {(4,5); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (3,1);
       (4,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (2,4); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (3,1);
       (4,1); (2,0); (3,0); (4,0)},
      {(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (4,1);
       (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (3,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (4,1);
       (2,0); (3,0); (4,0)},
      {(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (4,1); (2,0);
       (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (4,1); (2,0);
       (3,0); (4,0)},
      {(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (5,2); (6,2); (3,1); (2,0); (3,0);
       (4,0)}) /\
   SolitaireTrans
     ({(4,5); (4,4); (5,4); (6,4); (0,3); (1,3); (2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (5,2); (6,2); (3,1); (2,0); (3,0);
       (4,0)},
      {(4,5); (5,4); (6,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0);
       (4,0)}) /\
   SolitaireTrans
     ({(4,5); (5,4); (6,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0); (4,0)},
      {(4,5); (4,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,5); (4,4); (0,3); (1,3); (2,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0); (4,0)},
      {(0,3); (1,3); (2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(0,3); (1,3); (2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (2,0); (3,0); (4,0)},
      {(0,3); (1,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(0,3); (1,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (2,0); (3,0); (4,0)},
      {(2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (2,0); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (2,0); (3,0); (4,0)},
      {(2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(2,3); (4,3); (5,3); (6,3); (0,2); (1,2); (2,2); (3,2); (4,2); (5,2); (6,2); (3,1); (3,0); (4,0)},
      {(4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(4,3); (5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0); (4,0)},
      {(5,3); (6,3); (0,2); (1,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0); (4,0)}) /\
   SolitaireTrans
     ({(5,3); (6,3); (0,2); (1,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0); (4,0)},
      {(5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0)}) /\
   SolitaireTrans
     ({(5,3); (6,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0)},
      {(4,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0)}) /\
   SolitaireTrans
     ({(4,3); (0,2); (1,2); (3,2); (4,2); (5,2); (6,2); (2,1); (3,1); (3,0)},
      {(0,2); (1,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0)}) /\
   SolitaireTrans
     ({(0,2); (1,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0)},{(2,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0)}) /\
   SolitaireTrans ({(2,2); (3,2); (5,2); (6,2); (2,1); (3,1); (4,1); (3,0)},{(3,2); (5,2); (6,2); (3,1); (4,1); (2,0); (3,0)}) /\
   SolitaireTrans ({(3,2); (5,2); (6,2); (3,1); (4,1); (2,0); (3,0)},{(3,2); (4,2); (3,1); (4,1); (2,0); (3,0)}) /\
   SolitaireTrans ({(3,2); (4,2); (3,1); (4,1); (2,0); (3,0)},{(5,2); (3,1); (4,1); (2,0); (3,0)}) /\
   SolitaireTrans ({(5,2); (3,1); (4,1); (2,0); (3,0)},{(5,2); (3,1); (4,1); (4,0)}) /\
   SolitaireTrans ({(5,2); (3,1); (4,1); (4,0)},{(4,2); (5,2); (3,1)}) /\ SolitaireTrans ({(4,2); (5,2); (3,1)},{(3,2); (3,1)}) /\
   SolitaireTrans ({(3,2); (3,1)},{(3,3)}) /\ SolitaireFinal {(3,3)} : thm
```