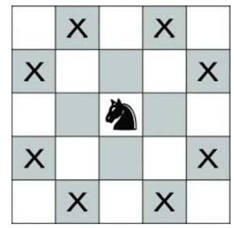
CS:4350 Logic in Computer Science

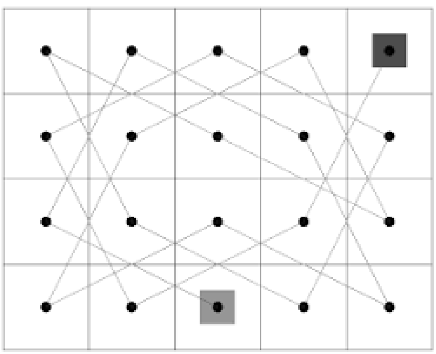
Midterm 2

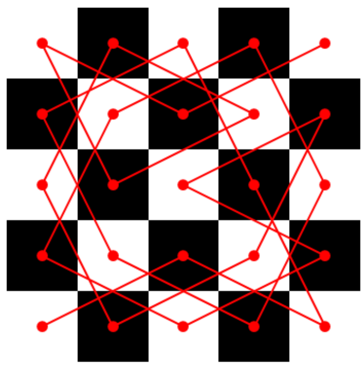
Dec 17, 2020, 8-11pm

(100 points)

On a chessboard, a knight can move up to eight other positions as shown in the figure. The Knight Tour problem is to find a sequence of knight's consecutive moves so that each position of the board is visited exactly once. Here the board can be of any dimension. For example, you may find below a solution of the 4x5, 5x5, and 8x8 board, respectively. Note that no solutions exist for boards of size 3x3 or 4x4.







1	60	15	24	47	36	13	26
16	23	64	59	14	25	38	35
63	2	61	46	37	48	27	12
22	17	56	49	58	51	34	39
3	62	21	52	45	40	11	28
18	55	44	57	50	31	8	33
43	4	53	20	41	6	29	10
54	19	42	5	30	9	32	7

Let the board dimension be (m, n). A *position* of the board is a pair of integers x/y, where $1 \le x \le m$ and $1 \le y \le n$. A *solution* of the Knight's tour is a sequence of t positions, where t=mn:

$$a_1/b_1,\ a_2/b_2,\ \ldots,\ a_t/b_t$$

such that each position a_i/b_i appears exactly once in the sequence and for any two consecutive positions a_i/b_i and a_{i+1}/b_{i+1}, a knight can go from a_i/b_i to a_{i+1}/b_{i+1}. In the following questions, for simplicity, we assume the starting position is 1/1.

1. (50 points) Let the propositional variable $p_{x,y,z}$ be true iff the knight's position after z-1 moves is x/y, where $1 \le x \le m$, $1 \le y \le n$, and $1 \le z \le t=mn$. Thus, $p_{1,1,1}$ will be true because 1/1 is the starting position. Please provide **(a)** a set S of propositional clauses such that the models of S match the solutions of the Knight's tour; **(b)** the number of clauses and the total number of literals in S in terms of m and n; **(c)** the encoding function which converts $p_{x,y,z}$ into an integer to be used in the DIMACS format for S; **(d)** all models of S when m=3 and n=4; **(e)** a method to let a SAT solver to find different solutions by adding into S some new clauses of length O(mn), if the SAT solver can produce at most one model per call .

Answer: **(a)** [25 pts] There are 5 groups of clauses in S. We will use p(x,y,z) for $p_{x,y,z}$ for convenience of writing; and if p(x,y,z) is true, we say "position x/y takes time slot z".

(i) The legal moves of a knight: We will use a shorthand r(x, y) for $(1 \le x \le m \wedge 1 \le y \le n)$, whose values are easily decided when x and y are instantiated with values, so r(x,y)p(x,y,z) = p(x,y,z) if r(x,y) = 1 and r(x,y)p(x,y,z) = 0 if r(x,y)=0.
For $1 \le x \le m$, $1 \le y \le n$, and $1 \le z < mn$,
(¬p(x,y,z) | r(x-2,y-1)p(x-2,y-1,z+1) | r(x-2,y+1)p(x-2,y+1,z+1) | r(x-1,y-2)p(x-1,y-2,z+1) | r(x-1,y+2)p(x-1,y+2,z+1) | r(x+1,y-2)p(x+1,y-2,z+1) | r(x+1,y+2)p(x+1,y+2,z+1) | r(x+2,y-1)p(x+2,y-1,z+1) | r(x+2,y+1)p(x+2,y+1,z+1))
There are mn(mn-1) propositional clauses from this pattern of clauses and each clause contains 3-9 literals, so the total number of literals is bound by 9mn(mn-1), or $O((mn)^2)$.

(ii) At any time slot, at most one position can take that slot: For $1 \le x, u \le m$, $1 \le y, v \le n$, and $1 \le z \le mn$, if ($x \ne u$ or $y \ne z$), then (¬p(x,y,z) | ¬p(u,v,z)).
The number of propositional clauses is $(mn)^2(mn-1)$ and the total number of literals is $2(mn)^2(mn-1)$, because each clause has 2 literals.

(iii) Each position can take at most one time slot: For $1 \le x \le m$, $1 \le y \le n$, and $1 \le z < u \le mn$, (¬p(x,y,z) | ¬p(x,y,u)).
The number of propositional clauses is $(mn)^2(mn-1)/2$ and the total number of literals is $(mn)^2(mn-1)$, because each clause has 2 literals.

(iv) Each position must take at least one time slot: For $1 \le x \le m$, $1 \le y \le n$, and t=mn, (p(x,y,1) | p(x,y,2) | … | p(x,y,t)).
The number of clauses is mn and each clause contains mn literals, so the total number of literals is $(mn)^2$. Note that this is the set of positive clauses; without this group, there is a trivial model by setting each variable false.

(v) The initial position: (p(1,1,1)). One clause and one literal.

(b) [5 pts] Adding all the clause numbers in (a), we get $O((mn)^3)$. The total number of literals is also $O((mn)^3)$.

(c) [5 pts] code(x,y,z) = ((x-1)*n + (y-1))*m*n + z for $1 \le x \le m$, $1 \le y \le n$, and $1 \le z \le mn$. Thus, code(1,1,1) = 1, and code(m,n,mn) = $(mn)^2$. For example, when m=3 and n=4, code(3,4,12) = 12^2 = 144.

(d) [10 pts] There are two models: (false variables are not displayed)
M_1 = { p(1,1,1), p(2,3,2), p(3,1,3), p(1,2,4), p(2,4,5), p(3,2,6), p(1,3,7), p(3,4,8), p(2,2,9), p(1,4,10), p(3,3,11), p(2,1,12) }
M_2 = { p(1,1,1), p(2,3,2), p(3,1,3), p(1,2,4), p(2,4,5), p(3,2,6), p(1,3,7), p(2,1,8), p(3,3,9), p(1,4,10), p(2,2,11), p(3,4,12) }

(e) [5 pts] Once a model is found, add the negation of all true variables in the model as one clause into the clause set S. . For example, the negation of M_1 in (d) is
$(\neg p(1,1,1) \mid \neg p(2,3,2) \mid \neg p(3,1,3) \mid \neg p(1,2,4) \mid \neg p(2,4,5) \mid \neg p(3,2,6) \mid \neg p(1,3,7) \mid \neg p(3,4,8) \mid \neg p(2,2,9) \mid \neg p(1,4,10) \mid \neg p(3,3,11) \mid \neg p(2,1,12))$
The same model cannot be found by the SAT solver and this clause contains mn literals since the number of true variables in each model of S is mn.

2. (50 points) Given the board dimension (m, n), let sol(L) be true iff L is a list of positions as a solution of the Knight Tour problem. A *move* is a relation among positions: move(a, b, c, d) is true iff a knight can move from position a/b to position c/d in a chessboard of size (m, n). Please **(a)** define sol(L) and move(a, b, c, d) in the first-order logic. You may use arithmetic operators and constants such as <, ≤, =, +, -, *, 0, 1, etc., without a definition. If you use other predicates, you must provide their definitions. **(b)** Write a Prolog program to solve the Knight Tour problem based on the first-order formulas in (a) and to display the solution as a sequence of positions.

Answer: **(a)** [30 pts]

∀x,y,z,u move(x,y,z,u) ↔ (1 ≤ x ≤ m ∧ 1 ≤ y ≤ n ∧ 1 ≤ z ≤ m ∧ 1 ≤ u ≤ n ∧ ((z=x-2 ∧ u=y+1) | (z=x-2 ∧ u=y+1) | (z=x-1∧ u=y-2) | (z=x-1∧ u=y+2) | (z=x+1∧ u=y-2) | (z=x+1∧ u=y+2) | (z=x+2∧ u=y-1) | (z=x+2∧ u=y+1)))

∀L sol(L) ↔ initialPosition(L) ∧ distinct(L) ∧ length(L)= m*n ∧ knightMoves(L)

In the following, we use Prolog list notations, where [] = nil, [X|Y] = cons(X, Y):

∀X initialPosition([1/1 | X])

∀X,Y distinct([]) ∧ distinct([X]) ∧
(distinct([X|Y]) ↔ (¬member(X,Y) ∧ distinct(Y)))

∀X,Y,L ¬member(X, []) ∧ (member(X, [Y|L]) ↔ (X=Y | member(X, L)))

length([]) = 0 ∧ ∀X,Y length([X|Y]) = length(Y)+1

∀X,Y,Z,U,L knightMoves([]) ∧ knightMoves([X]) ∧
(knightMoves([X/Y, Z/U | L]) ↔ (move(X,Y,Z,U) ∧ knightMoves([Z/U | L])))

The Prolog program below gives a more efficient implementation of sol(L) and move(a,b,c,d).

(b) [20 pts] Prolog code:

```
dimension(3, 4).   % definition of the board size

% sol(X) succeeds if X is a solution of the Knights tour for a board of dimension(M, N).
sol(X) :-
    dimension(M, N),            % get the dimension of the board
    generate_positions(M, N, L), % generate all positions of the board.
    select(1/1, L, L1),         % remove the initial position.
    T is M*N-1,                 % number of moves
    sol2(T, 1/1, L1, X).        % look for T moves from the initial position 1/1

% sol2(T, P, L, S) succeeds if L contains T positions and a knight
% can visit all positions of L once from the current position P consecutively,
% and S is the list of positions in the order of moves.
sol2(0, P, [], [P]).                 % the current position P is the last position in solution.
sol2(T, X/Y, L, [X/Y | R]) :-       % get partial solution R and add X/Y in the beginning
    T>0, T1 is T-1,                 % more steps to go
    select(X1/Y1, L, L1),           % pick the next position from the candidate list L
    move(X, Y, X1, Y1),             % from X/Y to X1/Y1 is a knight's move
    sol2(T1, X1/Y1, L1, R).         % get partial solution from the remaining positions.

% move(X, Y, X1, Y1) succeeds if a knight can move from X/Y to X1/Y1.
move(X, Y, X1, Y1) :- X>2, Y>1, X1 is X-2, Y1 is Y-1.
move(X, Y, X1, Y1) :- X>2, dimension(_, N), Y<N, X1 is X-2, Y1 is Y+1.
move(X, Y, X1, Y1) :- X>1, Y>2, X1 is X-1, Y1 is Y-2.
move(X, Y, X1, Y1) :- X>1, dimension(_, N), Y<N-1, X1 is X-1, Y1 is Y+2.
move(X, Y, X1, Y1) :- dimension(M, _), X<M, Y>2, X1 is X+1, Y1 is Y-2.
move(X, Y, X1, Y1) :- dimension(M, N), X<M, Y<N-1, X1 is X+1, Y1 is Y+2.
move(X, Y, X1, Y1) :- dimension(M, _), X<M-1, Y>1, X1 is X+2, Y1 is Y-1.
move(X, Y, X1, Y1) :- dimension(M, N), X<M-1, Y<N, X1 is X+2, Y1 is Y+1.

% generate(M, N, L) succeeds if L is a complete list of pairs x/y, 1<=x<=M, 1<=y<=N.
generate_positions(M, N, L) :-
    gen_list(1, M, X),
    gen_list(1, N, Y),
    gen_pairs(X, Y, L).

% gen_list( X, N, R) succeeds if R = [X, X+1, X+2, ..., N].
gen_list( X, N, []) :- X>N, !.
gen_list( X, N, [X | Res]) :- Y is X+1, gen_list( Y, N, Res).
```

```
% gen_pairs(X, Y, L) succeeds if X=[a1, ..., am], Y=[b1, ..., bn] and L = [a1/b1, am/bn]
gen_pairs( [], Y, []).
gen_pairs( [A|X], Y, Z) :- pairs(A, Y, R), gen_pairs(X, Y, Res), append(R, Res, Z).

% pairs(X, Y, R) succeeds if Y=[b1, ..., bn] and R=[X/b1, ..., X/bn].
pairs(X, [], []).
pairs(X, [Y | L], [X/Y | R]) :- pairs(X, L, R).
```

Bonus Points (5 points): list typos in the textbook.