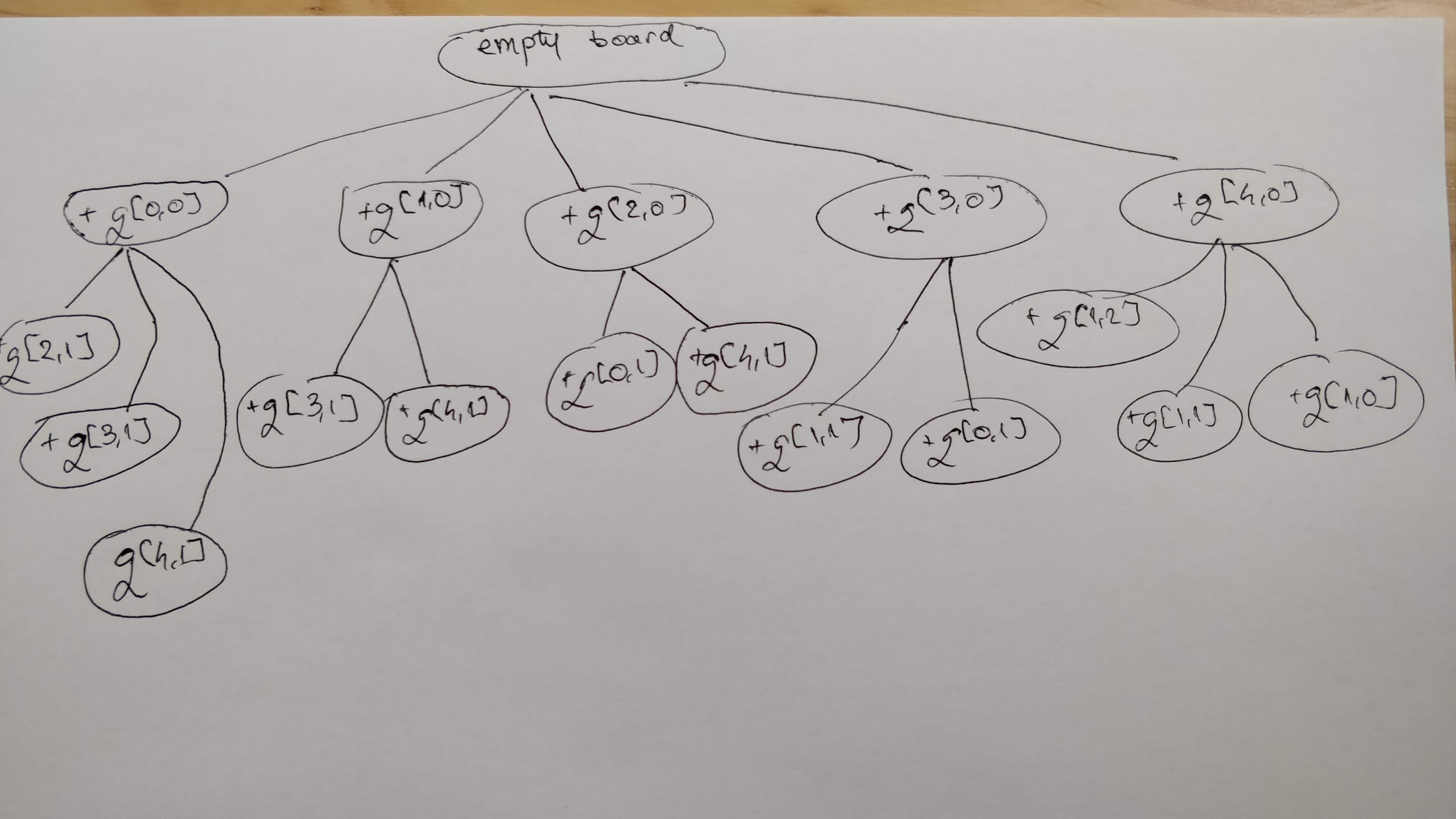
1a)

i)

* **states**: All arrangements of 5 queens, one per column, in the leftmost n ( 0 <= n <= 5) columns of the board, with no queen attacking another.
* **initial** **state**: the empty board
* **actions:** Add a queen to any empty square in the leftmost empty column such that it is not attacked by other queen
* **transition model:** returns the board with the queen added at the specified position
* **goal test:** 5 queens on the board, none is attacked.3

ii)

(Note: +q[X, Y] reads as add a queen at position (row X, column Y).

The tree should contain states (board configurations), so to construct them walk the tree from the root to the chosen node, adding a queen to the board built so far, at the specified position by each visited node.)

iii) Record the number of queens on each row (N). If N > 1 then the first queen on the chosen row is attacked by N - 1 queens. If N < 2, there are no conflicts/ attacks.

For each row, let a(row) be the result recorded above.

The heuristic function is the sum of all the records: h = a(r), and 1 <= h <= n, so it’s admissible.

In other words, h = 𝚺 (number of queens in a row - 1), for each row.

b) Ts1 = Ts({}) = {s(1), s(2), q(1, 1), q(2, 2)}

Ts2 = {s(1), s(2), q(1, 1), q(2, 2), r(1), r(2), }

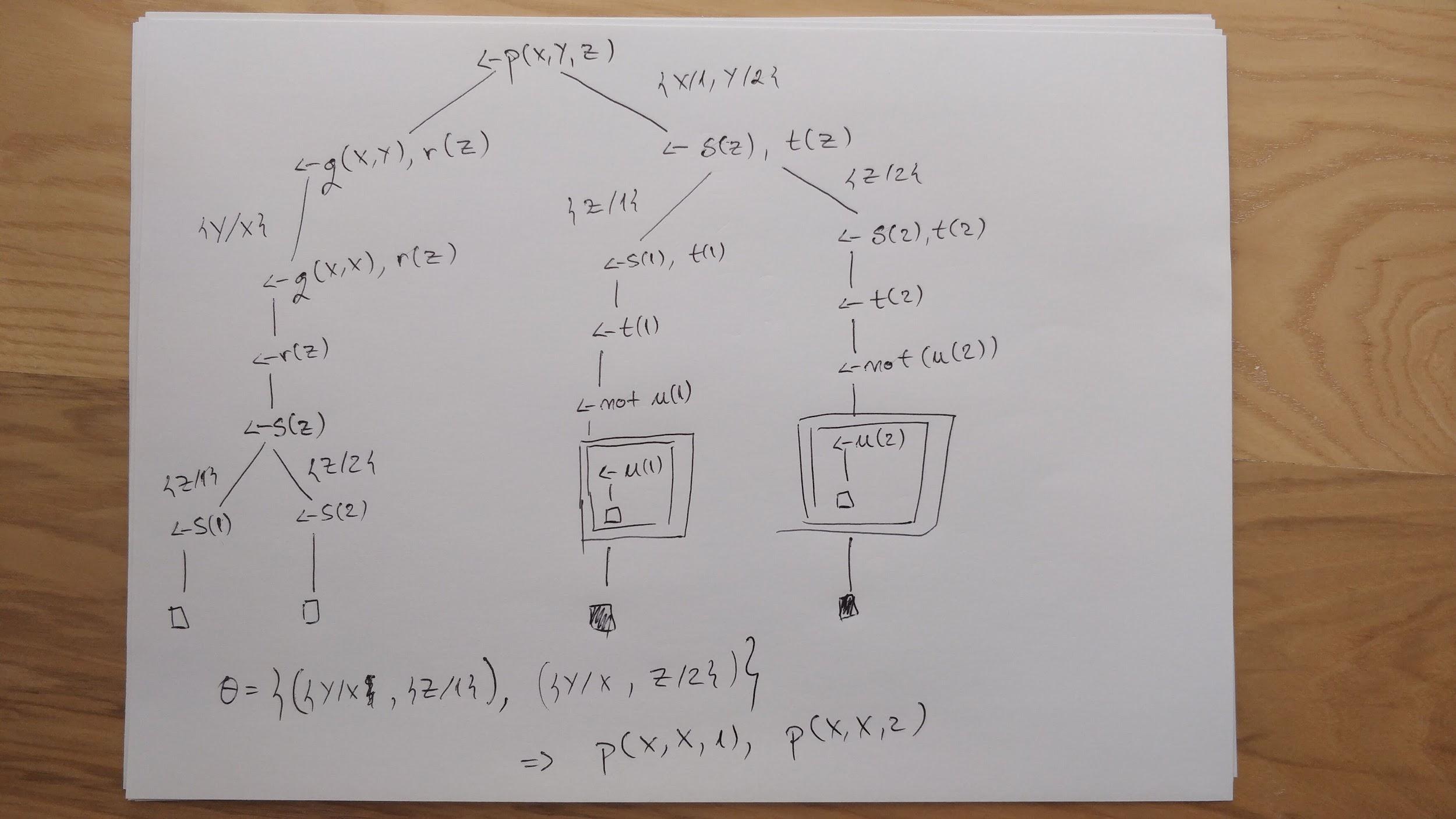
Ts3 = Ts2 U {p(1, 1, 1), p(1, 1, 2), p(2, 2, 1), p(2, 2, 2)}

Ts4 = Ts3

So LHM(S)=Ts3={s(1), s(2), q(1, 1), q(2, 2), r(1), r(2), p(1, 1, 1), p(1, 1, 2), p(2, 2, 1), p(2, 2,2)}

c)

i)



ii) u(X) ↔ T

q(X, X) ↔ T

s(X) ↔ X = 1 V X = 2

r(Z) ↔ s(Z)

t(X) ↔ ⌐ u(X)

p(X, Y, Z) ↔ (q(X, Y) ⋀ r(Z)) V (X = 1 ⋀ Y = 2 ⋀ s(Z) ⋀ t(Z))

All solutions are consistent with the Completion of S’ because there exists an SLDNF-refutation of p from S’ with answer θ.

ra

d) “a logical agent operates by deducing what to do from a knowledge base of sentences about the world. The knowledge base is composed of axioms—general knowledge about how the world works—and percept sentences obtained from the agent’s experience in a particular world.” (‘AI: A Modern Approach’ textbook page 265)

====================================

Edit: I think “a system that acts rationally” would do here, since this is the terminology used in the slides.

2a)

i)

initially(at(0, 0)).

initially(locked(1, 1)).

ic :- happens(E1, T), happens(E2, T), E1 =/= E2

position((0, 0)). position((0, 1)). position((1, 0)). position((1, 1)).

terminates(move(0, 1), locked(1, 1), T).

initiates(move(0, 1), unlocked(1, 1), T).

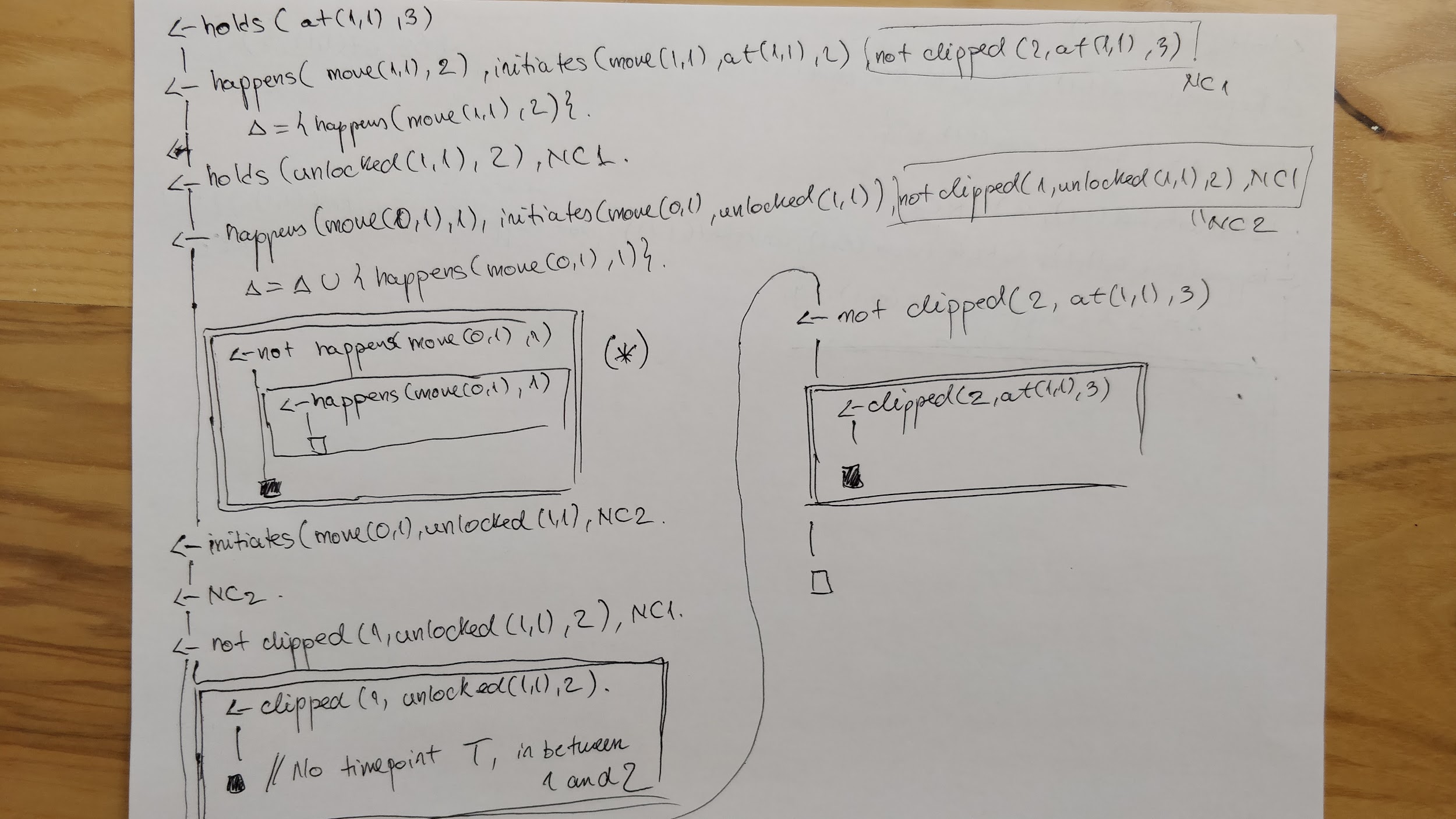
initiates(move(P), at(P), T) :-

holds(unlocked(P), T),

position(P).

ii) The set of abducibles A is defined as

**A** = {happens(A, T) | for all legal actions A and timepoints T defined in the problem’s domain}



The consistency check (\*) should be performed for the first abducible as well.

another version of 2a (works with abductive\_engine.pl):

ic :- happens(E1, T), happens(E2, T), E1 =/= E2.

initially(locked(1,1)).

initially(at(0,0)).

key(0,1).

cell(0,0). cell(0,1). cell(1,0). cell(1,1).

terminates(moveTo(X, Y), locked(A, B), T) :-

cell(A, B),

cell(X, Y),

key(X, Y).

initiates(moveTo(X, Y), at(X, Y), T) :-

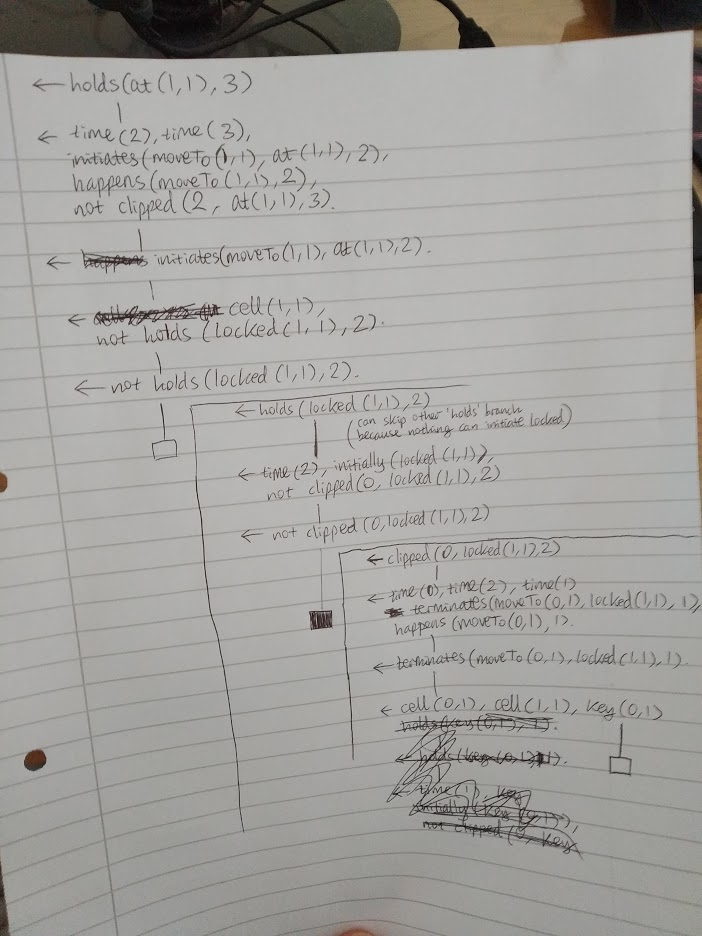
cell(X, Y),

\+ holds(locked(X, Y), T).

terminates(moveTo(A, B), at(X, Y), T) :-

cell(X, Y),

cell(A, B).



2b)

i) more or less the same as 2a) i) but with all the predicates ground + timepoints + explicitly defined set of abducibles

(Would the following be an example rule then?:)

initiates(moveTo(X, Y), at(X, Y), T) :- holds(unlocked(X,Y), T).

2ai but with the goal like:

:- not holds(at(1,1), 3).

and with the abducibles defined as a choice rule like:

0 { happens(moveTo(0,0),1..2);

happens(moveTo(0,1),1..2);

happens(moveTo(1,0),1..2);

happens(moveTo(1,1),1..2) } 8 (or infinity).

not super sure though

ii) The minimal model of its reduct should be equal to Δ

iii) :~ happens(move(X, Y), T) . [1@1, X, Y, T]. (?????)

or :~position(P), happens(move(P), T) . [1@1, P, T]

or :~ happens(A, T). [1@1, A, T]

iv) :- time(T), 2#count{X, Y: happens(move(X,Y), T)}. (??)

or maybe

:- 2#count{X, Y : happens(moveTo(X, Y), T)}, time(T), cell(X, Y).

Or :- 2#count(T, X, Y: holds(at(X, Y), T)).

:- position(X, Y), 2#count(T: happens(moveTo(X, Y), T)) ??