1 What predicates need to be implemented:

Predicates that need to be done

1. `satisfy(+Goals, +State)`
2. `satisfyGoal(+State)`
3. `op_Applicable(+State, ?OpName, ?Params)`
4. `op_ApplyOp(+OldState, +Step, ?NewState)`
5. `fluentsOutFrom(+Conditions, -Fluents)`

1. `satisfy(+Goals, +State)`

   Goals is a list of *Goal*. A *State satisfy Goals* iff *State satisfies* every *Goal* in *Goals*.

   State satisfyGoal iff *State satisfy* every goal in the problem's Goal list. The problem's Goal list is stored in problem. To access it, just execute "*problem(_, _, Goals)*" and *Goals* will be bound to the problems Goal list.

2. `satisfyGoal(+State)`

   This is similar to satisfy/2, except that this predicate specifically is asking whether the problem's goal is satisfied by *State*. This predicate needs to look up the problem's goal.

3. `op_Applicable(+State, ?OpName, ?Params)`

   OpName with *Params* is `op_Applicable` iff the *Preconditions* for *OpName* bound with the *Params* binding list makes *satisfy(Preconditions, State)* true.

4. `op_ ApplyOp(+OldState, +Step, ?NewState)`

   NewState is `op_Apply` to *OldState* via *Step* iff using *NewState* = *OldState* - *Step*'s negative *Effects* + *Step*'s positive *Effects*. In other words, The step's negated effects are deleted from *OldState* and the positive effects are added to form *NewState*.

   For example, if *OldState* = *[at(a), unvisited(b), unvisited(c)]*, the negative effects = *[not(at(a)), not(unvisited(b)]*, and the positive effects = *[at(b)]* then *NewState* = *[at(b), unvisited(c)]*. Note that even though the negative efects had *not* around them, it was their enclosed predicate that was deleted, e.g., *not(at(a))* deleted *at(a)* from the *NewState* list. Also note that *NewState* must be an ordered set, i.e., passes the `is_ordset(+Term)` test (see *ordset* library documentation).

5. `fluentsOutFrom(+Conditions, -Fluents)`

   Described in Agenda 1.

2 test cases for unit testing

For all the test cases below, you need to load both the problemSolver file (e.g., call [ld]) and the tsp domain files (e.g., call loadDomain(tsp, p02, hZero)).

1. `satisfy(+Goals, +State)`

   ?- A = 2, B = 3, satisfy([neq(A,B), not(at(a)), at(b), edge(a,b, _), not(edge(b,c,_))], [at(b)]).

   this should succeed with binding list A = 2, B = 3.

   This tests whether metaLevel preds, negative fluent preds, positive fluent preds, positive static preds, and negative static preds can succeed. You also need to check whether they can fail.

2. `satisfyGoal(+State)`

   ?- satisfyGoal([at(a)]).

   Should succeed.

   ?- satisfyGoal([at(b)]).

   Should fail.

3. `op_Applicable(+State, ?OpName, ?Params)`

   ?- =$op_{Applicable}$=([at(a), unVisited(a), unVisited(b)], OpName, Params).

   Should succeed with two different binding lists:

   - *OpName = selfVisit, Params = [a] ;*
   - *OpName = move, Params = [a, 2, b] ;*

4. `op_ ApplyOp(+OldState, +Step, ?NewState)`

   ?- `op_Applicable([at(a),` unVisited(a), unVisited(b)], OpName, Params), `op_record(OpName,` Params, _Preconds, _Effects, Cost), `make_step([opName(OpName),` opParams(Params), stepCost(Cost)], Step), `op_ApplyOp([at(a),` unVisited(a), unVisited(b)], Step, NewState).

   This should succeed with the binding list:

   OpName = selfVisit,
   Params = [a],
   _Preconds = [at(a), unVisited(a)],
   _Effects = [not(unVisited(a))],
   Cost = 0,
   Step = step(selfVisit, [a], 0),
   NewState = [at(a), unVisited(b)]

5. `fluentsOutFrom(+Conditions, -Fluents)`

   / ?- fluentsOutFrom([at(a), unVisited(a), unVisited(b), edge(a,b,2), edge(b,a,1)], Fluents)./

This should succeed with the binding list:
Fluents = [at(a), unVisited(a), unVisited(b)].