```
% GENERATE
% okres harmonogramowania [s]
time(1..3600). % 60 minut

% przypisac satelite do wykonywanej akcji, dla kazdej akcji
% obserwacja - moze byc wykonana na jednej z orbit z ktorej widac dany obszar
{ execute(Action,O) : orbit(O), visible(Action,O,1) } 1 :- action(Action,observe).
% downlink - moze, ale nie musi byc wykonane
{ execute(Action,O) : orbit(O), visible(Action,O,1) } :- action(Action,downlink).
% uplink - musi byc wykonane przez wszystkie satelity majace mozliwosc wykonania
execute(Action,O) :- orbit(O), visible(Action,O,1), action(Action,uplink).

% wybrac z mozliwych czas wykonywania zadan dla okna kazdej akcji, dla kazdego satelity
% czas wykonywania akcji musi miescic sie w dostepnych oknach czasowych
1 { action_scheduled(Action,O,Start,Start+Duration-1) : time(Start),
Start >= WStart,
Start+Duration-1 <= WEnd,
sat_action_time(O,Type,Duration),
action(Action,Type),
action_type(Type),
action_window(Action,WStart,WEnd)
 } 1 :- execute(Action,O).



% DEFINE
% parametry satelity
% zuzycie pamieci satelit w danym czasie
% downlink zwalnia zuzyta pamiec
sat_memory(O,Action,0) :- execute(Action,O), action(Action,downlink), orbit(O).
% obserwacje zajmuja pamiec
sat_memory(O,Action,Use) :- execute(Action,O), action(Action,observe), orbit(O),
LastDownlink = #max {A: execute(A,O), action(A,downlink), S=O, A < Action },
Use = #sum {Units,A,D,S: execute(A,O), action(A,observe), memory_use(O,Units), A <= Action, A >
D, S=O, D=LastDownlink }.

% zuzycie energii
sat_energy(O,Action,Amount) :- execute(Action,O), satellite(O,Sat),
sat_energy(O,0,Initial), energy_gen(O,Units), action_scheduled(Action,O,Start,End),
Cost = #sum { U,A : execute(A,Or), A <= Action, energy_use(Or,U), Or=O },
Gen = Units * Start,
Amount = Initial+Gen-Cost,
Amount <= Initial.
% maksymalne naladowanie akumulatora
```

```
sat_energy(O,Action,Max) :- sat_energy(O,Action,Amount), Amount > Max,
energy_storage(O,Max).

% statystyki dot. ilosci akcji wykonywanych przez satelity
executed(S,Cnt) :- satellite(O,S), Cnt = #count {A: execute(A,O) }.
observed(S,Cnt) :- satellite(O,S), Cnt = #count {A: execute(A,O), action(A,observe) }.
downlinked(S,Cnt) :- satellite(O,S), Cnt = #count {A: execute(A,O), action(A,downlink) }.


% TEST
% akcje nie moga sie na siebie nakladac
:- time(Time), orbit(O), satellite(O,Sat), #count { Action : execute(Action,O),
action_scheduled(Action,O,Start,End), Time>=Start, Time<=End} > 1.

% wykluczyc modele z przekroczeniem pamieci
:- sat_memory(O,Action,Val), memory_storage(O,Max), action(Action,Type), orbit(O), Val > Max.

% wykluczyc modele z wyczerpaniem energii
:- sat_energy(O,Action,Val), action(Action,Type), orbit(O), Val < 1.

% wykluczyc modele, w ktorych nie wykonano istniejacych zadan awaryjnych
:- not execute(Action,_), action(Action,observe), emergency_task(Action).


% OPTIMISE
% jakosc harmonogramowania zadan (suma priorytetow przypisanych zadan)
quality(X) :- X = #sum { Priority,Action : priority(Action,Priority), execute(Action,O) }.

% maksymalizacja jakosci harmonogramowania
#maximize { X@5 : quality(X) }.

%% DISPLAY
#show action_scheduled/4.
#show executed/2.
#show observed/2.
#show downlinked/2.
#show quality/1.
#show sat_memory/3.
#show sat_energy/3.
#show action_window/3.
#show execute/2.
```