CS 6610: Software Engineering II

Example
The farmer and all his possessions will be represented as Objects. Some objects eat other objects when the Farmer's not around.

abstract sig Object { eats: set Object }

one sig Farmer, Fox, Chicken, Grain e
- Module
  models/examples/tutorial/farmer
- open util/ordering[State]
/* * Define what eats what when the Farmer isn't around. * Fox eats the chicken and the chicken eats the grain. */

fact eating { eats = Fox->Chicken +
/* * The near and far relations contain the objects held on each * side of the river in a given state, respectively. */

sig State { near: set Object, far: set Object }
/* * In the initial state, all objects are on the near side. */

fact initialState { let s0 = first() | s0.near = Object &&& no s0.far }
/* * Constrains at most one item to move from 'from' to 'to'. * Also constrains which objects get eaten. */

pred crossRiver (from, from', to, to': set Object) {  // either the Farmer takes no items (from' = from - Farmer &&& to' = to - to.eats + Farmer ) || // or the Farmer takes one item some item: from - Farmer { from' = from - Farmer - item to' = to - to.eats + Farmer + item } }
/* * crossRiver transitions between states *
 */ fact stateTransition { all s: State, s': next(s) { Farmer in s.near => crossRiver(s.near, s'.near, s.far, s'.far), crossRiver(s.far, s'.far, s.near, s'.near) } } /*
/* * the farmer moves everything to the far side of the river. */
pred solvePuzzle () { last().far = Object } run solvePuzzle for 8 State expect 1