```
% Author:  Bruce Spencer
% Date: 4/7/2012
% cip is condor in prolog, a very rough prototype to gain
experience with the condor system

% Thea is used to load OWL ontologies so we can handle
them as a set of axioms.
% :- assertz(library_directory('c:/users/spencerb/thea')).
%
:- use_module(library(thea2/owl2_io)). %Assumes thea2 is
on the library path
run(OWLFileIn) :-
   load_axioms(OWLFileIn, owl),
   cip_axioms.

%cipAxioms changes the unions and intersections to formulae

cipAxioms :-
   retractall(cipSubClassOf(_,_)),
   subClassOf(C, D),
   convertUnionsIntersections(C, C1),
   convertUnionsIntersections(D, D1),
   assert(cipSubClassOf(C1, D1)),
   fail.
cipAxioms :-
   equivalentClasses(L), %Thea's equivalentClasses are
always pairs
   append(_, [C | L1], L), member(D, L1),
   convertUnionsIntersections(C, C1),
   convertUnionsIntersections(D, D1),
   assert(cipSubClassOf(C1, D1)),
   assert(cipSubClassOf(D1, C1)),
   fail.
cipAxioms:-
   disjointClasses(L),
   append(_, [C | L1], L), member(D, L1),
   convertUnionsIntersections(C, C1),
   convertUnionsIntersections(D, D1),
   complement(D1, D1C),
   assert(cipSubClassOf(C1, D1C)),
   fail.
```

```
cipAxioms.

convertUnionsIntersections(Formula, FC):-
    Formula = intersectionOf(Cs),
    convertIntersection(Cs, FC).
convertUnionsIntersections(Formula, FC):-
    Formula = unionOf(Cs),
    convertUnion(Cs, FC).
convertUnionsIntersections(Formula, FC):-
    Formula = someValuesFrom(R, C),
    convertUnionsIntersections(C, C1),
    FC = someValuesFrom(R, C1).
convertUnionsIntersections(Formula, FC):-
    Formula = allValuesFrom(R, C),
    convertUnionsIntersections(C, C1),
    FC = allValuesFrom(R, C1).
convertUnionsIntersections(Formula, FC):-
    Formula = complementOf(C),
    convertUnionsIntersections(C, C1),
    FC = complementOf(C1).
convertUnionsIntersections(C, C):-
    class(C).

convertUnion([C1, C2], unionOf(C1C, C2C)):-
   convertUnionsIntersections(C1, C1C),
   convertUnionsIntersections(C2, C2C).
convertUnion([C1, C2, C3 | CR], unionOf(C1C, CRU)):-
   convertUnionsIntersections(C1, C1C),
   convertUnion([C2, C3 | CR], CRU).

convertIntersection([C1, C2], intersectionOf(C1C, C2C)):-
   convertUnionsIntersections(C1, C1C),
   convertUnionsIntersections(C2, C2C).
convertIntersection([C1, C2, C3 | CR], intersectionOf(C1C,
CRU)):-
   convertUnionsIntersections(C1, C1C),
   convertIntersection([C2, C3 | CR], CRU).

complement(complementOf(C), C):- !.
complement(C, complementOf(C)).
```

```
complementarySign(pos, neg).
complementarySign(neg, pos).

% Polarity of Occurrence
polarityOfOccurrence(C, C, pos) :-
   classFormula(C).
polarityOfOccurrence(C, Formula, Sign) :-
   (
    Formula = intersectionOf(C1, _);
    Formula = intersectionOf(_, C1);
    Formula = unionOf(C1, _);
    Formula = unionOf(_, C1);
    Formula = someValuesFrom(_, C1);
    Formula = allValuesFrom(_, C1);
    Formula = subClassOf(_, C1)
   ),
   polarityOfOccurrence(C, C1, Sign).
polarityOfOccurrence(C, Formula, Sign) :-
   (
    Formula = subClassOf(C1, _D);
    Formula = complementOf(C1)
    ),
   polarityOfOccurrence(C, C1, OtherSign),
   complementarySign(Sign, OtherSign).

classFormula(Formula) :-
    class(Formula);
    Formula = intersectionOf(_, _);
    Formula = unionOf(_, _);
    Formula = someValuesFrom(_, _);
    Formula = allValuesFrom(_, _).
polarityCheck(Class, subClassOf(C, D), Sign) :-
   cipSubClassOf(C, D),
   polarityOfOccurrence(Class, subClassOf(C, D), Sign).


%Structure Transformation
stTrans :-
   retractall(structTransSubClassOf(_)),
   cipSubClassOf(C, D),
   polarityOfOccurrence(Class, subClassOf(C, D), Sign),
```

```
   (class(Class) ->
       STClass = Class;
    Class = complementOf(CompClass), class(CompClass) ->
       with_output_to(atom(CompClassName),
write(CompClass)),
       STClass = complementOf(CompClassName);
    Class = intersectionOf(C1, D1) ->
       with_output_to(atom(C1Name), write(C1)),
       with_output_to(atom(D1Name), write(D1)),
       STClass = intersectionOf(C1Name, D1Name);
    Class = unionOf(C1, D1) ->
       with_output_to(atom(C1Name), write(C1)),
       with_output_to(atom(D1Name), write(D1)),
       STClass = unionOf(C1Name, D1Name);
    Class = someValuesFrom(R, C1) ->
       with_output_to(atom(C1Name), write(C1)),
       STClass = someValuesFrom(R, C1Name);
    Class = allValuesFrom(R, C1) ->
       with_output_to(atom(C1Name), write(C1)),
       STClass = allValuesFrom(R, C1Name);
    %else raise an alert
       writef('CIP Structural Transformation: Unknown
structure %w\n', [Class])
    ),
    with_output_to(atom(ClassName), write(Class)),
    (Sign = pos ->
        assert(structTransSubClassOf(ClassName, STClass));
     %Sign = neg
        assert(structTransSubClassOf(STClass, ClassName))
     ),
     fail.
stTrans :-
    cipSubClassOf(C, D),
    with_output_to(atom(CName), write(C)),
    with_output_to(atom(DName), write(D)),
    assert(structTransSubClassOf(CName, DName)),
    fail.
stTrans.
```