**Assignment 1**

**Semantic Networks & Frames**

*Due date: 9th Nov. 2020.*

**Imagine you are to give a lecture on Semantic Networks and Frames.**

**Create a powerpoint presentation for this lecture.**

The presentation should include:

1. An original (i.e. created by you) example of a Semantic Network. The Semantic Network should have enough detail to answer what follows and, yet, be small enough for a newcomer to easily follow.
2. A detailed explanation of Semantic Networks.
3. Examples demonstrating all the properties of Semantic Networks.
4. The representation of part 1-3 using a system of Frames.
5. A detailed explanation of Frames.
6. Examples demonstrating all the properties of Frames.
7. Prolog code implementing your system of Frames. If you use the code printed on the following pages then include the comment “% Bratko Code V1”, “% Bratko Code V2” etc. Also, state that in your presentation. If you have modified Bratko’s code then write “% Modified Bratko Code V1” etc. Modified code should then be explained in the presentation. Similarly, if you have written your own code then write “% My Code” and explain it in your presentation.
8. The explanation of your Prolog code. Any code that is written by you should be thoroughly explained i.e. its purpose, sample questions that use this code and a trace of how prolog answers these questions.
9. The output of the code plus explanation (This must be the output of SWI Prolog). The properties you demonstrated in part 6 should be included in the output i.e. in the presentation give examples of questions that might be asked and answers that were given.

Note: You should create a single sample problem that is rich enough to demonstrate all properties of both Semantic Networks and Frames. If your sample problem can’t be used to demonstrate a property then you should, at least, demonstrate it with a different sample problem.

**Take note:**

Do not share any part of your work – all work should be original (see CA Rules).

**What you are to do:**

1. You should **email** the following to [shane.dowdall@dkit.ie](mailto:shane.dowdall@dkit.ie):

1. A SWI Prolog file called **your\_names.pl** (e.g. shane\_dowdall\_joe\_bloggs.pl) which contains the prolog program that you created.
2. A Powerpoint file called **your\_names.pptx**

2. You should also **print a copy** of the Powerpoint file, the Prolog file, and an enlarged version of the Semantic Network picture and hand them up along with a signed copy of the “**Assignment Cover Sheet**”.

Note 1: You should not bind the print out – just staple the sheets together at top left hand corner.

Note 2: There will be a penalty (a reduction of 10% per week) for late submissions.

Note 3: There will be a penalty (a reduction of 10%) for not handing up the requested print outs.

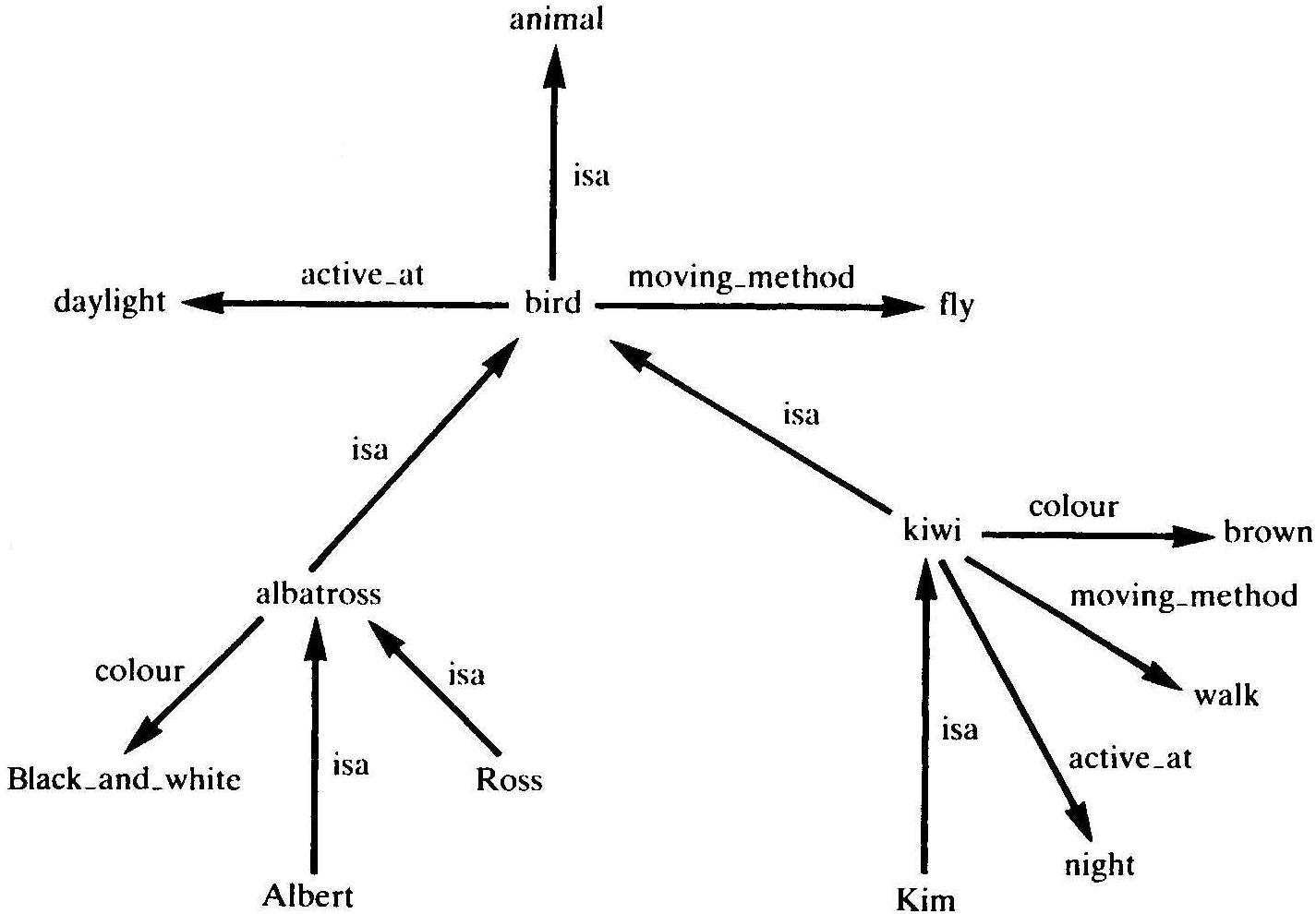
**How you may work:**

You should work in pairs. Both individuals will be given the same project mark.

Possible implementation of

Semantic Networks and Frames

Section 15.7, PROLOG Programming for Artificial Intelligence ed. 3, Ivan Bratko.



**Implementation of this Sematic Network using PROLOG:**

% implementation of Sematic Network using PROLOG

isa( bird, animal).

isa( albatross, bird).

isa( ross, albatross).

isa( kim, kiwi).

moving\_method( bird, fly).

moving\_method( kiwi, walk).

**% Code V1**

fact( Fact) :- % Fact not a variable; Fact = Rel( Arg1, Arg2)

Fact,!. % Fact explicit in network - do not inherit

fact( Fact) :-

Fact =.. [ Rel, Arg1, Arg2],

isa( Arg1, SuperArg), % Climb isa hierarchy

SuperFact =.. [ Rel, SuperArg, Arg2],

fact( SuperFact).

**Sample Questions:**

?- fact( moving\_method( bird, Method) ).

?- fact( moving\_method( kim, Method) ).

?- fact( moving\_method( ross, Method) ).

**Exercise:**

**Convert the above Semantic Network into a series of frames.**

**Writing these as Frames in PROLOG:**

% A frame is represented as a set of Prolog facts:

% frame\_name( Slot, Value)

% where Value is either a simple value or a procedure

% Frame bird: the prototypical bird

bird( a\_kind\_of, animal).

bird( moving\_method, fly).

bird( active\_at, daylight).

% Frame albatross: albatross is a typical bird with some

% extra facts: it is black and white, and it is 115 cm long

albatross( a\_kind\_of, bird).

albatross( colour, black\_and\_white).

albatross( size, 115).

% Frame kiwi: kiwi is a rather untypical bird in that it

% walks instead of flies, and it is active at night

kiwi( a\_kind\_of, bird).

kiwi( moving\_method, walk).

kiwi( active\_at, night).

kiwi( size, 40).

kiwi( colour, brown).

% Frame albert: an instance of a big albatross

albert( instance\_of, albatross).

albert( size, 120).

% Frame ross: an instance of a baby albatross

ross( instance\_of, albatross).

ross( size, 40).

**The information held in these frames can be accessed using the following code:**

**% Code V2**

value( Frame, Slot, Value) :-

Query =.. [ Frame, Slot, Value],

call( Query),!.

value( Frame, Slot, Value) :-

parent( Frame, ParentFrame),

value( ParentFrame, Slot, Value).

parent( Frame, ParentFrame) :-

(Query =.. [Frame, a\_kind\_of, ParentFrame]

;

Query =.. [Frame, instance\_of, ParentFrame]),

call(Query).

**Sample Questions:**

?- value( albert, active\_at, A).

?- value( kiwi, active\_at, K).

**To allow for calculated slots we need to alter the previous program as follows:**

% Frames with calculated slots

% A frame is represented as a set of Prolog facts:

% frame\_name( Slot, Value)

% where Value is either a simple value or a procedure

% Frame bird: the prototypical bird

bird( a\_kind\_of, animal).

bird( moving\_method, fly).

bird( active\_at, daylight).

% Frame albatross: albatross is a typical bird with some

% extra facts: it is black and white, and it is 115 cm long

albatross( a\_kind\_of, bird).

albatross( colour, black\_and\_white).

albatross( size, 115).

% Frame kiwi: kiwi is a rather untypical bird in that it

% walks instead of flies, and it is active at night

kiwi( a\_kind\_of, bird).

kiwi( moving\_method, walk).

kiwi( active\_at, night).

kiwi( size, 40).

kiwi( colour, brown).

% Frame albert: an instance of a big albatross

albert( instance\_of, albatross).

albert( size, 120).

% Frame ross: an instance of a baby albatross

ross( instance\_of, albatross).

ross( size, 40).

% Frame animal: slot relative\_size obtains its **Note: from here down is different**

% value byexecuting procedure relative\_size

animal( relative\_size, execute( relative\_size( Object, Value), Object, Value) ).

**% Code V3** **Note: all this code is different**

value( Frame, Slot, Value) :-

value( Frame, Frame, Slot, Value).

value( Frame, SuperFrame, Slot, Value) :-

Query =.. [ SuperFrame, Slot, Information],

call( Query),

process( Information, Frame, Value),!.

value( Frame, SuperFrame, Slot, Value) :-

parent( SuperFrame, ParentSuperFrame),

value( Frame, ParentSuperFrame, Slot, Value).

process( execute( Goal, Frame, Value), Frame, Value) :- !,

call( Goal).

process( Value, \_, Value).

parent( Frame, ParentFrame) :-

(Query =.. [Frame, a\_kind\_of, ParentFrame]

;

Query =.. [Frame, instance\_of, ParentFrame]),

call(Query).

relative\_size( Object, RelativeSize) :-

value( Object, size, ObjSize),

value( Object, instance\_of, ObjClass),

value( ObjClass, size, ClassSize),

RelativeSize is ObjSize/ClassSize \* 100.

**Sample Questions:**

?- value( albert, active\_at, A).

?- value( kiwi, active\_at, K).

?- value( ross, relative\_size, R).

**Note the approach that Bratko has taken to implementing Semantic Networks and Frames. You should ask yourself whether this implementation satisfies all your requirements. What are its limitations? E.g. how is multiple inheritance implemented?**

**Prolog Code “ ! ”**

! – no backtracking to find alternative answer once first solution found.

**Prolog Code “ =.. ”**

Predicate for decomposing terms and constructing new terms: **=.. reads as “univ”**

The goal: **Term =..L**

Is true if **L** is a list that contains the principal functor of **Term**, followed by its arguments.

The following examples illustrate:

**?- f( a, b) =..L.**

**L = [f, a, b]**

**?- T =..[ rectangle, 3, 5].**

**T = rectange( 3, 5)**

**?- Z =..[p, X, f( X, Y)].**

**Z = p( X, f( X, Y) )**

In some programs we may have code such as:

# Goal =..[Functor| Arglist], …

and we wish to invoke the result for execution. This is obtained by using **call**:

# Goal =..[Functor| Arglist],

call( Goal).

For example if we type the following into **family tree.pl**:

**question(X,Y,Z) :-**

**Q =.. [ X, Y, Z],**

**call(Q).**

we can then ask the question: **?- question( pred, X, Y).**

which is the same as asking the question: **?- pred( X, Y).**

Similarly, if we type the following into **family tree.pl**:

bq( X, F, A) :-

X =.. [F, A].

And then ask the following: **?- bq( male(john), F, A).**

we get the result **F = male,**

**A = john**

We can use lists by typing the following into **family tree.pl:**

bq( X, F, A) :-

X =.. [F|A].

And then ask the following: **?- bq( pred( X, Y), F, A).**

we get the result **F = pred,**

**A = [X, Y] ;**