Sherlock

"Eliminate all other factors, and the one which remains must be the truth."

Author:
Alexander Brown

Module:
CS26210

Assignment:
A Chatbot in Prolog

April 28, 2012

Contents

Chapter 1

Design

1.1 Architecture

The overall architecture of "Sherlock" is somewhat inspired by Internation Business Machines (IBM) Watson. However due to the private nature of that code I can't completely base my design on Watson; a good choice due to the need for "Sherlock" to run without the aid of a mainframe.

Another design choice I'm keen to take is to use the Resource Description Framework (RDF), for which SWI-Prolog has an existing and supporting library.

The beauty of RDF is that it has a predicate-based triplet structure, which would make Input data for "Sherlock" very easy to both load and produce.

With time constraints it's unreasonable to expect "Sherlock" to learn over time as many natural-language based systems tend to. As such AI conecepts such as Neural Networks are also unreasonable.

With this in mind, what can be reasonably expected from "Sherlock"?

1. Understand what the question is looking for.
2. Search a 'database' of information for a given subject.
3. Respond with a sensible answer.

Of the three, searching and responding are the simple modules; Natural Language processing being renound for being a complex field of AI.

1.2 RDF Database

RDF are based on the idea of expressing simple statments with resources, where each statement consists of a subject, a predicate and an object.

http://www.example.org/index.html has a **creator** whose value is **John Smith**

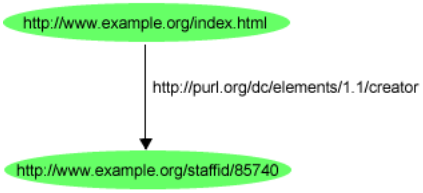

Figure 1.1: A Simple RDF Statement[1]

RDF gets interesting when you consider joining objects. Nodes (subjects or objects) are connected using arcs (predicates). Nodes can be joined together and chain down. Using the example given by the specification tadpole could be represented by Figure 1.2.

So how does this help? Well Prolog works with predicates which are also an integral part of RDF. So we can make logical connection in our database of RDFs.

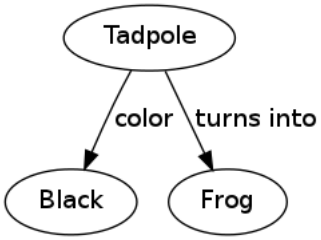

Figure 1.2: An exaple of how the tadpole example could be defined using the RDF graph structure.

The other neat part of RDFs is that it takes away a number of prolems associated with the natural language processing as the predicates are a already in place and defined by the RDFs (colour, turns into, etc.).

The final reason for going down an RDF route is the extensibility of their structure; potentially any data could be represented in the RDF database whilst being able to keep the base.

1.3 Natual Language Processing

With the use of RDF databases to model data there's very little need to do any actual natural language processing. That is to say, the question can be inferred from the words input and their existence in RDF data types, with the inclusion of a database of synonyms (which can also be store in an RDF structure).

To solve this problem somewhat, answers from a synonym are reduced in weight (the answer being returned as a confidence level, much like IBM Watson does).

To futher this approach the predicates of the found topics should be returned and analysed to find any common themes and format it into the answer.

Chapter 2

The Nature of Understanding in the Context of Predicate Logic

2.1 What is Understanding?

Understanding is a complex subject because it is a hard concept to grasp; for a start knowledge does not imply understanding and vice versa. One can have knowledge of all the parts of an engine and yet not understand how those pieces fit together as a whole. On the flipside one can understand how engines work but have no knowledge of the specific parts of an engine.

The Chinese Room is a thought experiment presented by John Searle and has very close ties to computational understanding.

The overall principal of the Chinese Room is that a black box recieves messages in chinese, processes them using a set of rules and gives a response in chinese. The operator of the box has no external contact (aside from the messages they recieve) and has no understanding of chinese.

The black box is evidently a metaphor for a computer (computers being very good at performing large sets of instructions). The operator can appear to have a brilliant understanding of chinese depending on how encompassing the rules are.

However this brings about a interesting conundrum, as the human mind only works using electrical signals - much as a computer does. So can the human mind truely be replicated by a computer?

Given that we don't understand the full nature of the human mind yet, one cannot say for sure. And the lines become even futher blurred when you add that a lot of computers can 'learn', quite similar to how humans do. Again I return to IBM Watson which processes a huge amount of training data before it can produce any useful answers. It can even learn from the mistakes it has made in the past so that it can only improve.

So is this really understanding? Not entirely I would say. One thing all these knowledge-based systems cannot do at the moment is use their current knowledge and invent their own knowledge. They may be able to uncover knowledge that has not been spotted before, even optimise existing solutions. But they currently do not make the distrinctions between their many knowledge areas to improve overall knowledge of both. Then, and only then, would I start considering reevaluating my position on whether computers can ever truely understand anything.

2.2 "Sherlock" and Understanding

Sherlock's understanding comes from the input RDF data, which is, at the end of the day, produced by humans. Not computers.

I make no bones that Sherlock cannot understand the subject it represents, whatever that may be. However a part of Sherlock is about being able to understand the question it is being asked, understanding what is required for a response.

Chapter 3

Testing

```
Script started on Sat 28 Apr 2012 20:32:29 BST
softly@edea ~/dev/sherlock $ ./sherlock
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.10.4)
Copyright (c) 1990-2011 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- ask(['are','tadpoles','black']). % Should pass.
I have the upmost confidence that is true (99.90% confidence).
true.

?- ask(['are','tadpoles','dark']). % Should pass, but with less confidence
I am rather certain that is true (89.91% confidence)
true.

?- ask(['are','tadpoles','green']). % Should fail.
I do not believe it is likely (0.00% confidence)
false.

?- ask(['do','tadpoles','turn','into','frogs']). % Should pass
I am rather certain that is true (89.91% confidence)
true.

?- halt. % That's all there is defined in the RDF database.
?- % However this could go a lot futher with more data.
softly@edea ~/dev/sherlock $ exit

Script done on Sat 28 Apr 2012 20:34:05 BST
```

Chapter 4

Evaluation

Due to a lack of time due to work commitments, "Sherlock" isn't as complete as it could be. For example there is currently a bug where it can be 200% confident in a given answer and not being able to respond with anything other than a confidence level when it should be able to return some sort of idea of what the question was about.

Most of these issues would actually be easily fixed. The bug with 200% confidence is likely an issue which requires use of the cut (`!`) operator. The not being able to associate subject with the question (and answer) would be solved by being able to store the predicates from the RDF data.

Chapter 5

Code

5.1 ask.pl

```
% Main program file. Main asking routines stored here.
:- module(ask, [ask/1]).
:- use_module(['./rdf_utils.pl', './synonym.pl']).

ask:-
    read(Question),
    ask(Question).

% ask(+Input, -Answer, -Total) - Ask a question.
%     Input  - list of strings.
%     Answer - The results found.
%     Total  - The number of RDF references found.
%
%     Will print out it's confidence of the question being true.
ask(Input):-
    ask(Input, Answer, Total),
    printResults(Answer, Total).

% printResults(+Answer, +Total) - Special case for no Total.
%     Will print a statement for when there is no knowledge in the RDF
%     database about any subject).
printResults(_, 0):-
    format('I\'m afraid I don\'t know anything about that topic.~n',[]),

    % Cut to stop backtracking, then fail.
    !,
    fail.

% printResults(+Answer, +Total) - Print results.
%     Answer - The results found.
%     Total  - The number of RDF references relating to topics any
%              subject in the question.
%
%     Will create a percentage value and then print results based on
%     that percentage.
printResults(Answer, Total):-
    Confidence is (Answer/Total)*100,
    printResults(Confidence).
```

```
% printResults(+Confidence) - Print results for a confidence level of
%                             90% or more.
%     Confidence - The percentage confidence level.
printResults(Confidence):-
    Confidence >= 90,
    format('I␣have␣the␣upmost␣confidence␣that␣is␣true␣(~2f%␣confidence).~n',
        [Confidence]),
    !.

% printResults(+Confidence) - Print results for a confidence level of
%                             75% or more.
%     Confidence - The percentage confidence level.
printResults(Confidence):-
    Confidence >= 75,
    format('I␣am␣rather␣certain␣that␣is␣true␣(~2f%␣confidence)~n', Confidence),
    !.

% printResults(+Confidence) - Print results for a confidence level of
%                             50% or more.
%     Confidence - The percentage confidence level.
printResults(Confidence):-
    Confidence >= 50,
    format('I␣am␣somewhat␣sure␣of␣that␣(~2f%␣confidence)~n', Confidence), !.

% printResults(+Confidence) - Print results for any other confidence
%                             level (i.e. not confident at all).
%     Confidence - The percentage confidence level.
printResults(Confidence):-
    format('I␣do␣not␣believe␣it␣is␣likely␣(~2f%␣confidence)~n', Confidence),
    !, % Stop backtracking and fail.
    fail.

% ask(+Input, -Answer, -Total) - Entry point for ask/6.
%     Input  - The input array.
%     Answer - The results.
%     Total  - The number of RDF references found.
ask(Input, Answer, Total):-
    ask(Input, Input, 0, 0, Answer, Total).

% ask(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) - End point
%                                                  for ask/6.
%     Array   - The array being read from. Empty.
%     Input   - The array to recurse into. Unused.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Will take the input Answer and Total scores (IAnswer and ITotal)
%     and set them to the outputs (OAnser, OTotal).
ask([], _, IAnswer, ITotal, OAnswer, OTotal):-
    OAnswer is IAnswer,
```

```
    OTotal is ITotal.

% ask(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) - Calls
%                                          ask_subject/4 and recurses.
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Ensures there is an RDF reference for the next element in Array
%     and goes calls into ask_subject/4
ask([Subject|Tail], Input, IAnswer, ITotal, OAnswer, OTotal):-
    has_rdf(Subject),
    ask_subject(Subject, Input, IAnswer, NOAnswer),
    !, % Don't allow backtracking as ITotal will changed.
    ask(Tail, Input, NOAnswer, ITotal + 1, OAnswer, OTotal).

% ask(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) - Finds
%                                          synonyms and searchs on those.
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Calls into ask_synonym/6 to check for plurals and synonyms.
ask([Subject|Tail], Input, IAnswer, ITotal, OAnswer, OTotal):-
    has_synonym(Subject),
    synonym(Subject, Synonyms),
    ask_synonym(Synonyms, Input, IAnswer, ITotal, NOAnswer, NOTotal),
    !, % No backtracking as things may changed.
    ask(Tail, Input, NOAnswer, NOTotal, OAnswer, OTotal).

% ask(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) - Add a
%                                          little to the total if
%                                          nothing was found.
%
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     If no RDF was found for the subject add a little to the total.
ask([Subject|Tail], Input, IAnswer, ITotal, OAnswer, OTotal):-
    format('~w not found~n', Subject),
    ask(Tail, Input, IAnswer, ITotal + 0.01, OAnswer, OTotal),
    !.
```

```
% ask_synonym(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       End point of ask_synonym/6.
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Set output to inputs.
ask_synonym([], _, IAnswer, ITotal, OAnswer, OTotal):-
    OAnswer is IAnswer,
    OTotal is ITotal.

% ask_synonym(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       Calls ask_subject/4 and recurses.
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Ensures there is an RDF reference for the next element in Array
%     and goes calls into ask_subject/4
ask_synonym([Subject|Tail], Input, IAnswer, ITotal, OAnswer, OTotal):-
    has_rdf(Subject),
    ask_subject(Subject, Input, IAnswer, NOAnswer), !,
    ask_synonym(Tail, Input, NOAnswer, ITotal + 1.001, OAnswer, OTotal).

% ask_synonym(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       Continuating
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
%     Recurse onwards.
ask_synonym([_|Tail], Input, IAnswer, ITotal, OAnswer, OTotal):-
    ask_synonym(Tail, Input, IAnswer, ITotal, OAnswer, OTotal),
    !.

% ask_synonym(+Array, +Input, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       Continuating
%     Array   - The array being read from.
%     Input   - The array to recurse into.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Ouput Total count.
%
```

```
%     Recurse onwards.
ask_synonym(Subject, Input, IAnswer, ITotal, OAnswer, OTotal):-
    ask_synonym([Subject], Input, IAnswer, ITotal, OAnswer, OTotal).

% ask_subject(+Subject, +Array, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       End point for ask_subject/6
%     Subject - The Subject to search for.
%     Array   - The Array to recurse into on ask_subject/6.
%     IAnswer - The Input Answer count.
%     ITotal  - The Input Total count.
%     OAnswer - The Output Answer count.
%     OTotal  - The Output Total count.
%
%     Will take the inputs and set them to the outputs for return values.
ask_subject(_, [], IAnswer, OAnswer):-
    OAnswer is IAnswer.

% ask_subject(+Subject, +Array, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       Works out confidences for the
%                                       current Subject.
%     Subject - The subject to search for.
%     Array   - The array to recurse into.
%     IAnswer - The input answer count.
%     ITotal  - The input total count.
%     OAnswer - The output answer count.
%     OTotal  - The output total count.
%
%     First of this checks that the Subject is not equal to the Input as
%     asking 'X is X?' is assumed to be true.
%
%     Then checks there is an RDF document.
%
%     Then calls are/4 to work out confidences.
%
%     Recurse with ITotal being incremented so that precents can be
%     generated.
%
%     Note: this should take into account synonyms and related words
%     (e.g. tadpole -> tadpoles)
ask_subject(Subject, [Input|Tail], IAnswer, OAnswer):-
    not_linked(Subject, Input),
    !,
    has_rdf(Subject),
    are(Subject, [Input], IAnswer, NOAnswer),
    !,
    ask_subject(Subject, Tail, NOAnswer, OAnswer).

% ask_subject(+Subject, +Array, +IAnswer, +ITotal, -OAnswer, -OTotal) -
%                                       Continuation.
%     Subject - The subject to search for.
%     Array   - The array to recurse into.
%     IAnswer - The input answer count.
%     ITotal  - The input total count.
```

```
%     OAnswer - The output answer count.
%     OTotal  - The output total count.
%
%     Continuation point for subjects that are the same as the input,
%     or which have no RDF document.
ask_subject(Subject, [_|Tail], IAnswer, OAnswer):-
    ask_subject(Subject, Tail, IAnswer, OAnswer).

% not_linked(+Word, +Other) - Ensures a word is not linked to another.
%     Word  - The word to ensure is not linked.
%     Other - The other word to check against.
%
%     Useful to ensure Sherlock is not checking a synonym is the same
%     as another.
%
%     This checks the synonyms.
not_linked(Word, Other):-
    synonym(Other, Synonyms),
    !,
    not_linked_synonym(Word, Synonyms).

% not_linked(+Word, +Other) - Ensures a word is not linked to another.
%     Word  - The word to ensure is not linked.
%     Other - The other word to check against.
%
%     Useful to ensure Sherlock is not checking a synonym is the same
%     as another.
not_linked(Word, Other):-
    %TODO - This may need to be above the aforementioned.
    Word \= Other.

% not_linked_sunonym(Word, Synonyms) - Ensures synonyms are not linked
%                                      to a word.
%     Word     - The word to ensure is not linked to the synonyms.
%     Synonyms - The list of synonyms.
%
%     This is the end point, which means they're not linked.
not_linked_synonym(_, []).

% not_linked_sunonym(Word, Synonyms) - Ensures synonyms are not linked
%                                      to a word.
%     Word     - The word to ensure is not linked to the synonyms.
%     Synonyms - The list of synonyms.
%
%     Call not_linked_synonym/2 with the singular word.
%
%     Sometimes there's only one synonym which doesn't get returned as
%     a list. Doing it this way allows Sherlock to deal with either
%     type easily.
not_linked_synonym(Word, [Other|_]):-
    not_linked_synonym(Word, Other).

% not_linked_sunonym(Word, Synonyms) - Ensures synonyms are not linked
```

```
%                                       to a word.
%       Word     - The word to ensure is not linked to the synonyms.
%       Synonyms - The list of synonyms.
%
%       Continuation point.
not_linked_synonym(Word, [_|Tail]):-
    not_linked_synonym(Word, Tail).

% not_linked_sunonym(Word, Synonyms) - Ensures synonyms are not linked
%                                       to a word.
%       Word  - The word to ensure is not linked to the synonyms.
%       Other - The other word to check against.
%
%       As not_linked/2.
not_linked_synonym(Word, Other):-
    Word \= Other.

% are(+Subject, +Array, +IAnswer, -OAnswer) - End point for are/4.
%       Subject - The subject currently being searched for.
%       Array   - The input array being recursed through.
%       IAnswer - The input answer count.
%       OAnswer - The output answer count.
%
%       Take IAnswer and assigns it to OAnswer for return values.
are(_, [], IAnswer, OAnswer):-
    OAnswer is IAnswer.

% are(+Subject, +Array, +IAnswer, -OAnswer) - Search RDF data.
%       Subject - The subject currently being searched for.
%       Array   - The input array being recursed through.
%       IAnswer - The input answer count.
%       OAnswer - The output answer count.
%
%       Search RDF data for references to the subject.
are(Subject, [Lookup|Tail], IAnswer, OAnswer):-
    get_rdf(Subject, RDFData),
    rdf_references(Lookup, RDFData, NoReferences), !,
    are(Subject, Tail, IAnswer + NoReferences, OAnswer).

% are(+Subject, +Array, +IAnswer, -OAnswer) - Search RDF data.
%       Subject - The subject currently being searched for.
%       Array   - The input array being recursed through.
%       IAnswer - The input answer count.
%       OAnswer - The output answer count.
%
%       Continuation for are/4.
are(Subject, [_|T], IAnswer, OAnswer):-
    are(Subject, T, IAnswer, OAnswer).
```

5.2 synonym.pl

```
% Search a special file for synonyms and plurals (related words).
:- module(synonym, [synonym/2, has_synonym/1]).
:- use_module(['./rdf_utils.pl', library(rdf)]).

% The subject of the synonym file.
subject('synonym').

% not(+X) - Logical not of X.
%     X - The call to negate.
not(X):-
    call(X),
    !,
    fail.

% has_synonym(+Word) - Ensures a Word has a synonym.
%     Word - The word to check has a synonym.
has_synonym(Word):-
    synonym(Word, List),
    !,
    not_empty(List).

% not_empty(+List) - Checks if a list is empty.
%     List - The list to check is empty.
not_empty([]):- fail.

% not_empty(+List) - Checks if a list is empty.
%     List - The list to check is empty.
not_empty(_).

% synonym(+Word, -Synonyms) - Get synonyms of a word.
%     Word     - The word to get synonyms of.
%     Synonyms - The synonyms of that word.
%
%     Ensure the synonym file can be loaded correctly.
synonym(_, _):-
    subject(X),
    not(has_rdf(X)).

% synonym(+Word, -Synonyms) - Get synonyms of a word.
%     Word     - The word to get synonyms of.
%     Synonyms - The synonyms of that word.
synonym(Word, Synonyms):-
    subject(X),
    get_rdf(X, RDFData),
    get_synonyms(Word, RDFData, Synonyms).

% get_synonym(+Word, -Synonyms) - Get synonyms of a word using RDF data.
%     Word     - The word to get synonyms of.
%     RDFData  - The RDF data.
%     Synonyms - The synonyms of that word.
%
```

```
%     Calls into get_synonyms/4 with an empty input.
get_synonyms(Word, RDFData, Synonyms):-
    get_synonyms(Word, RDFData, [], Synonyms).

% get_synonyms(+Word, +RDFData, -Input, +Synonyms) - End point for getting
%                                                    Synonyms.
%     Word     - The word to get synonyms of.
%     RDFData  - The RDF data of synonyms.
%     Input    - The loaded synonyms
%     Synonyms - The synonyms to return.
get_synonyms(_, [], Input, Synonyms):-
    append([], Synonyms, Input).

% get_synonyms(+Word, +RDFData, -Input, +Synonyms) - Load the synonyms.
%     Word     - The word to get synonyms of.
%     RDFData  - The RDF data of synonyms.
%     Input    - The loaded synonyms
%     Synonyms - The synonyms to return.
get_synonyms(Word, [rdf(Word, _, Value)|Tail], Synonyms, Output):-
    get_value(Value, Full),
    !,
    append(Synonyms, Full, NewSynonyms),
    get_synonyms(Word, Tail, NewSynonyms, Output).

% get_synonyms(+Word, +RDFData, -Input, +Synonyms) - Load synonyms.
%     Word     - The word to get synonyms of.
%     RDFData  - The RDF data of synonyms.
%     Input    - The loaded synonyms
%     Synonyms - The synonyms to return.
get_synonyms(Word, [rdf(Subject, _, Value)|Tail], Synonyms, Output):-
    get_value(Value, Word),
    !,
    append(Synonyms, Subject, NewSynonyms),
    get_synonyms(Word, Tail, NewSynonyms, Output).

% get_synonyms(+Word, +RDFData, -Input, +Synonyms) - Continuation.
%     Word     - The word to get synonyms of.
%     RDFData  - The RDF data of synonyms.
%     Input    - The loaded synonyms
%     Synonyms - The synonyms to return.
get_synonyms(Word, [_|Tail], Synonyms, Output):-
    get_synonyms(Word, Tail, Synonyms, Output).
```

5.3 rdf_utils.pl

```
% Utilities for RDF Documents.
:- module(rdf_utils, [has_rdf/1, get_rdf/2, rdf_references/3, get_value/2]).

% Use the main swipl RDF library.
:- use_module([library(rdf), './synonym.pl']).

% dir('./rdf/') - The location of the RDF database.
dir('./rdf/').

% extension('.rdf') - The RDF file extension.
extension('.rdf').

% get_file(+Subject, -File) - Convert a subject to an RDF data file.
%     Subject - The subject title to search for.
%     File    - The returned filename.
%
%     The formate is: dir(Dir) + Subject + extension(Ext).
%
%     Example: get_file('tadpole', File) sets File to './rdf/tadpole.rdf'.
get_file(Subject, File):-
    dir(Dir),
    extension(Ext),
    concat(Dir, Subject, Temp),
    concat(Temp, Ext, File).

% has_rdf(+Subject) - Checks if a subject has an RDF file.
%     Subject - The subject to check for an RDF document.
has_rdf(Subject):-
    get_file(Subject, File),
    exists_file(File),
    load_rdf(File, _).

% get_rdf(+Subject, -Data) - Gets the RDF data for the subject.
%     Subject - The subject to get the RDF document of.
%     Data    - The RDF data, produced by the RDF library.
get_rdf(Subject, Data):-
    has_rdf(Subject),
    get_file(Subject, File),
    load_rdf(File, Data).

% rdf_references(+Subject, +RDFData, -Refs) - Search for RDF references.
%     Subject - The subject to look for references in the RDF data.
%     RDFData - The RDF data to search through.
%     Refs    - The reference count.
%
%     Call rdf_references/4 with 0 as the input reference count.
rdf_references(Subject, RDFData, Refs):-
    rdf_references(Subject, RDFData, 0, Refs).

% rdf_references(+Lookup, +Data, +IRefs, -ORefs) - End point for the RDF
%                                                  refereces search.
```

```
%     Lookup - The lookup to search for references in.
%     Data   - The RDF data to search through.
%     IRefs  - The input reference count.
%     ORefs  - The output reference count.
rdf_references(_, [], IRefs, ORefs):-
    ORefs is IRefs,
    !.

% rdf_references(+Lookup, +Data,  +IRefs, -ORefs) - Search through RDF
%                                                   references search.
%     Lookup - The lookup to search for references in.
%     Data   - The RDF data to search through.
%     IRefs  - The input reference count.
%     ORefs  - The output reference count.
rdf_references(Lookup, [rdf(_, _, Value)|Tail], IRefs, ORefs):-
    value(Value, Lookup),
    !,
    rdf_references(Lookup, Tail, IRefs+1, ORefs).

% rdf_references(+Lookup, +Data,  +IRefs, -ORefs) - Search through RDF
%                                                   references search.
%     Lookup - The lookup to search for references in.
%     Data   - The RDF data to search through.
%     IRefs  - The input reference count.
%     ORefs  - The output reference count.
%
%     Checks through synonyms, as full references haven't been found.
%
%     This is weighted slightly lower to allow confidences to work fully.
rdf_references(Lookup, [rdf(_,_,Value)|Tail], IRefs, ORefs):-
    get_value(Value, RealValue),
    synonym(RealValue, Synonym),
    value(Lookup, Synonym),
    !,
    rdf_references(Lookup, Tail, IRefs + 0.9, ORefs).

% rdf_references(+Lookup, +Data,  +IRefs, -ORefs) - Continuation.
%     Lookup - The lookup to search for references in.
%     Data   - The RDF data to search through.
%     IRefs  - The input reference count.
%     ORefs  - The output reference count.
rdf_references(Lookup, [_|Tail], IRefs, ORefs):-
    rdf_references(Lookup, Tail, IRefs, ORefs).

% value(+Value, +Lookup) - Fail if no references were found.
%     Lookup - The word to search for.
%     Value  - The values from the RDF data.
value([],_):- fail.

% value(+Value, +Lookup) - Succede if the reference is a match.
%     Lookup - The word to search for.
%     Value  - The value from the RDF data. Equal to Lookup.
value(Lookup, Lookup).
```

```
% value(+Value, +Lookup) - Succede if the reference is a match to
%                          the RDF literal.
%     Lookup - The word to search for.
%     Value  - The value from the RDF data. Equal to literal(Lookup).
value(literal(Lookup), Lookup).

% value(+Value, +Lookup) - Downcase both sides as it shouldn't be
%                          case sensative.
%     Lookup - The word to search for.
%     Value  - The value from the RDF data.
value(literal(Value), Lookup):-
    downcase_atom(Value, LValue),
    downcase_atom(Lookup, LLookup),
    value(LValue, LLookup).

% value(+Values, +Lookup) - Search through the RDF values for referencess.
%     Lookup - The word to search for.
%     Values  - The values from the RDF data.
value([Value|_], Lookup):-
    value(Value, Lookup).

% value(+Values, +Lookup) - Search through the RDF values for referencess.
%     Lookup - The word to search for.
%     Values  - The values from the RDF data.
value([_|Tail], Lookup):-
    value(Tail, Lookup).

% get_value(+Values, -FullValue) - Get the value of RDF element.
%     Value      - The value from the RDF data.
%     FullValue  - The string value.
get_value(literal(Value), FullValue):-
    downcase_atom(Value, LValue),
    get_value(LValue, FullValue).

% get_value(+Values, -FullValue) - Get the value of RDF element.
%     Value      - The value from the RDF data.
%     FullValue  - The string value.
get_value(Value, FullValue):-
    append([], Value, FullValue).
```

Bibliography

[1] W3C. Rdf primer. http://www.w3.org/TR/rdf-primer/, March 2004.