Introduction to Logical English (LE) on SWISH - Version 1.0

02/11/2021
(Updates in red indicate material in draft. In blue, recent changes)

LE is syntactic sugar for pure Prolog. It has no knowledge of English, except for the meaning of the English logical connectives, if", "and", "or, "it is not the case that", etc, of the articles "a", "an" and "the", of the interrogative determiner "which" and of the relative pronoun "that" used to link a meta-predicate (or higher-order predicate) before "that" to another predicate after "that". LE also recognises the common nouns "day" and "date" as types, and checks that they are instantiated correctly.

LE has no knowledge of English grammar. It simply uses a list of words

fragment0 par1 fragement1 par2 fragment2... parn fragmentn

to represent a logical formula

fragment0_fragement1_fragment2._.. _fragmentn(par1, par2, …, parn)

where fragment0_fragement1_fragment2_... _fragmentn is a predicate symbol. Whereas predicate symbols are fixed constants, par1, par2, …, parn are parameters that can vary from one instance of the predicate to another.

LE is much more restrictive than most controlled natural languages. Whereas many controlled natural languages allow several ways to express logical quantification, for example:

Every bird is an animal.
All birds are animals.
If something is a bird then it is an animal.
Something is an animal if the thing is a bird.

LE has in effect only one way to do so, using the articles "a", "an" and "the":

a thing is an animal if the thing is a bird.

Once the name of the predicate has been identified by means of template declarations, the only variety possible is in the names of the variables. For example:

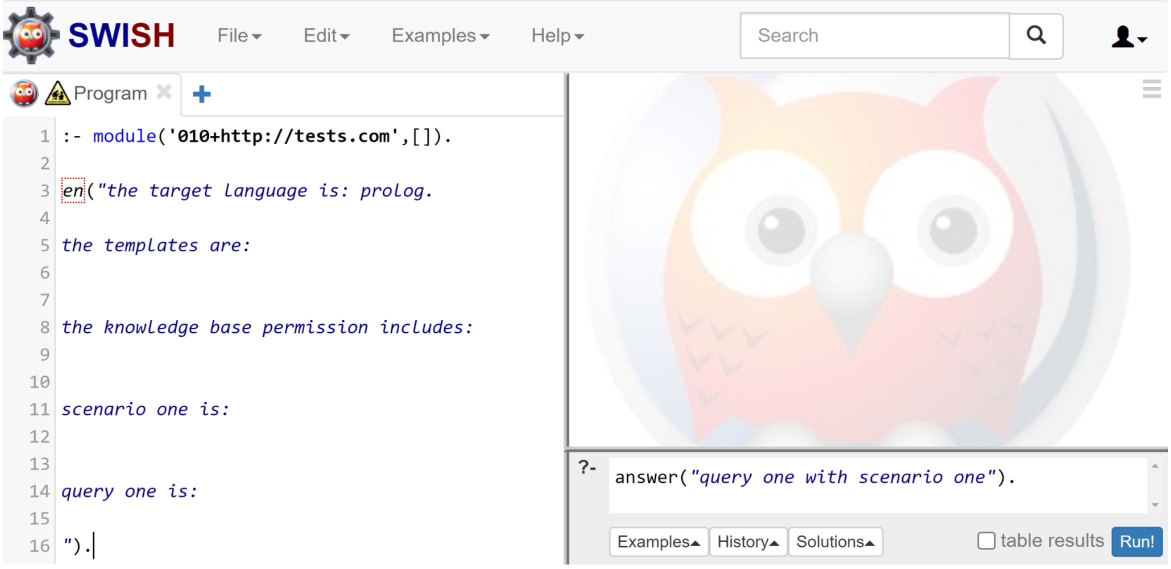
a thing is an animal if the thing is a bird.
an object is an animal if the object is a bird.
an individual is an animal if the individual is a bird.
an arbitrary list of words is an animal if the arbitrary list of words is a bird.

Prolog is an untyped language, and, except for “day” and “date”, the current version of LE is also untyped. This means that other common nouns, such as “animal”, “bird”, “thing”, “object”, etc, are simply meaningless mnemonic identifiers.

For more information about LE on SWISH, see: https://docs.google.com/document/d/1ATkGDBJN2eKzd681KNWrUwfrUI5MSguo/edit?usp=sharing&ouid=100982016581057551811&rtpof=true&sd=true

1. A simple example

A bare LE document on SWISH looks like this:



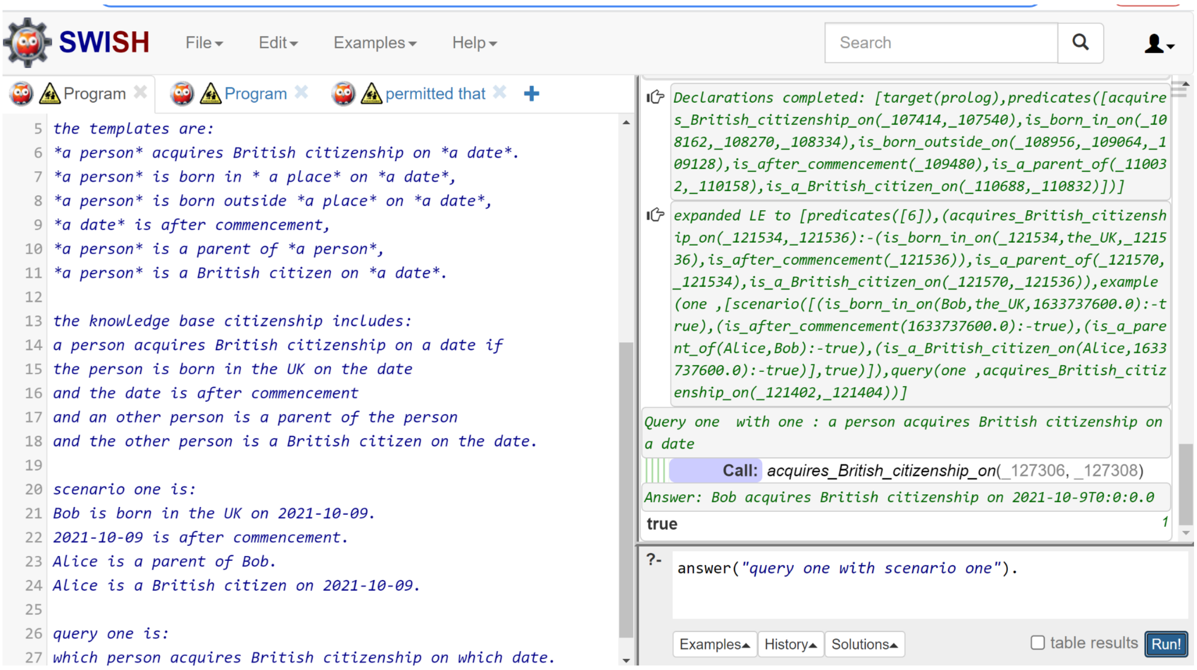
```
:- module('010+http://tests.com',[]).

en("the target language is: prolog.

the templates are:


the knowledge base permission includes:


scenario one is:


query one is:

").
```

```
?- answer("query one with scenario one").
```

The left pane contains a program consisting of a Prolog fact of the form en("…."). This prolog fact wraps the text of an LE document.

The bottom right pane contains a Prolog query, and the upper right pane contains the result of executing the query. Here is a simple LE program, scenario and query. The example can be found at

https://logicalenglish.logicalcontracts.com/p/citizenship.pl



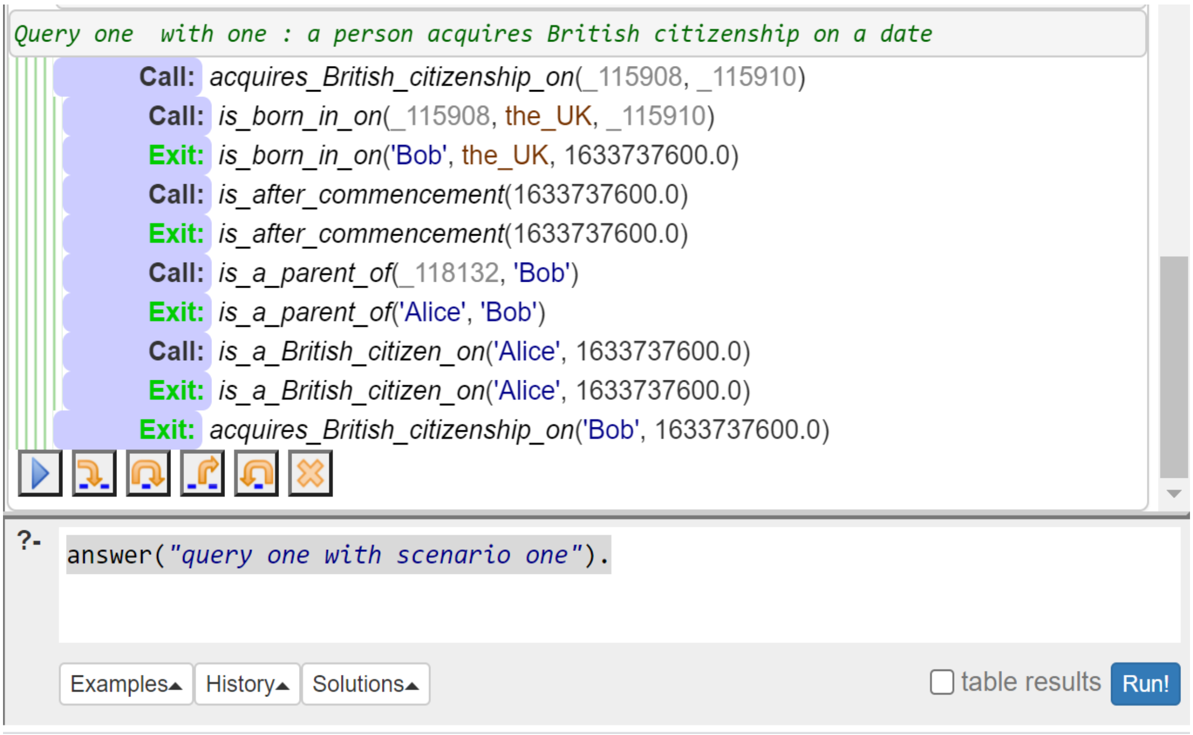
```
the templates are:
*a person* acquires British citizenship on *a date*.
*a person* is born in * a place* on *a date*,
*a person* is born outside *a place* on *a date*,
*a date* is after commencement,
*a person* is a parent of *a person*,
*a person* is a British citizen on *a date*.

the knowledge base citizenship includes:
a person acquires British citizenship on a date if
the person is born in the UK on the date
and the date is after commencement
and an other person is a parent of the person
and the other person is a British citizen on the date.

scenario one is:
Bob is born in the UK on 2021-10-09.
2021-10-09 is after commencement.
Alice is a parent of Bob.
Alice is a British citizen on 2021-10-09.

query one is:
which person acquires British citizenship on which date.
```

```
Declarations completed: [target(prolog),predicates([acquires_British_citizenship_on(_107414,_107540),is_born_in_on(_108162,_108270,_108334),is_born_outside_on(_108956,_109064,_109128),is_after_commencement(_109480),is_a_parent_of(_110032,_110158),is_a_British_citizen_on(_110688,_110832)])]

expanded LE to [predicates([6]),(acquires_British_citizenship_on(_121534,_121536):-(is_born_in_on(_121534,the_UK,_121536),is_after_commencement(_121536)),is_a_parent_of(_121570,_121534),is_a_British_citizen_on(_121570,_121536)),example(one ,[scenario([(is_born_in_on(Bob,the_UK,1633737600.0):-true),(is_after_commencement(1633737600.0):-true),(is_a_parent_of(Alice,Bob):-true),(is_a_British_citizen_on(Alice,1633737600.0):-true)],true)]),query(one ,acquires_British_citizenship_on(_121402,_121404))]

Query one  with one : a person acquires British citizenship on a date

Call: acquires_British_citizenship_on(_127306, _127308)

Answer: Bob acquires British citizenship on 2021-10-9T0:0:0.0

true

?- answer("query one with scenario one").
```

Programs can be saved and can be given a name in the file menu. However, it is important to save a separate backup.

The result of executing the Prolog command answer("query one with scenario one") is shown in the upper right pane. It displays an answer to the LE query, the Prolog debugger, and the Prolog predicates and clauses corresponding to the LE templates, knowledge base, scenarios and queries. There can be several LE scenarios and LE queries.

The display of Prolog clauses can be useful for debugging LE programs and queries, as can be the Prolog debugger itself:



```
Query one  with one : a person acquires British citizenship on a date
 Call: acquires_British_citizenship_on(_115908, _115910)
  Call: is_born_in_on(_115908, the_UK, _115910)
  Exit: is_born_in_on('Bob', the_UK, 1633737600.0)
  Call: is_after_commencement(1633737600.0)
  Exit: is_after_commencement(1633737600.0)
  Call: is_a_parent_of(_118132, 'Bob')
  Exit: is_a_parent_of('Alice', 'Bob')
  Call: is_a_British_citizen_on('Alice', 1633737600.0)
  Exit: is_a_British_citizen_on('Alice', 1633737600.0)
 Exit: acquires_British_citizenship_on('Bob', 1633737600.0)

?- answer("query one with scenario one").
```

2. Templates

A template represents a predicate. An instance of a template represents an atomic sentence[1], which can be true or false, and which can be combined with other atomic sentences in rules in a knowledge base.

A template is a list of words and parameters separated by spaces.
A word is a list of letters or numbers without spaces.
A parameters is a list of words prefixed by an asterix * and post-fixed by *.

The first word following a prefixed asterix must be one of the articles “a”, “an” or “the”. Although this restriction is not strictly necessary, it helps to check that the templates and their instances read naturally in English. On the other hand, except for the articles and for “day” and “date”, none of the words in a template have to be English. In particular, words can be symbols like X and Y.

The list of templates needs to be separated by commas or full stops.

In addition to the templates listed in the program, there are templates for Prolog built-in predicates and for Prolog defined predicates. The predefined LE templates include the infix form of some Prolog built-in binary predicates.

3. Template Instance

An instance of a template is obtained from the template by replacing every parameter of the template by a list of words separated by spaces. There need not be any relationship between the words in a parameter and the words in the instance of the parameter. Different parameters in the same template can be replaced by different or identical instances.

If the first word in the list replacing a parameter is one of the articles “a”, “an” or “the”, then the list represents a variable.

4. Meta-variables

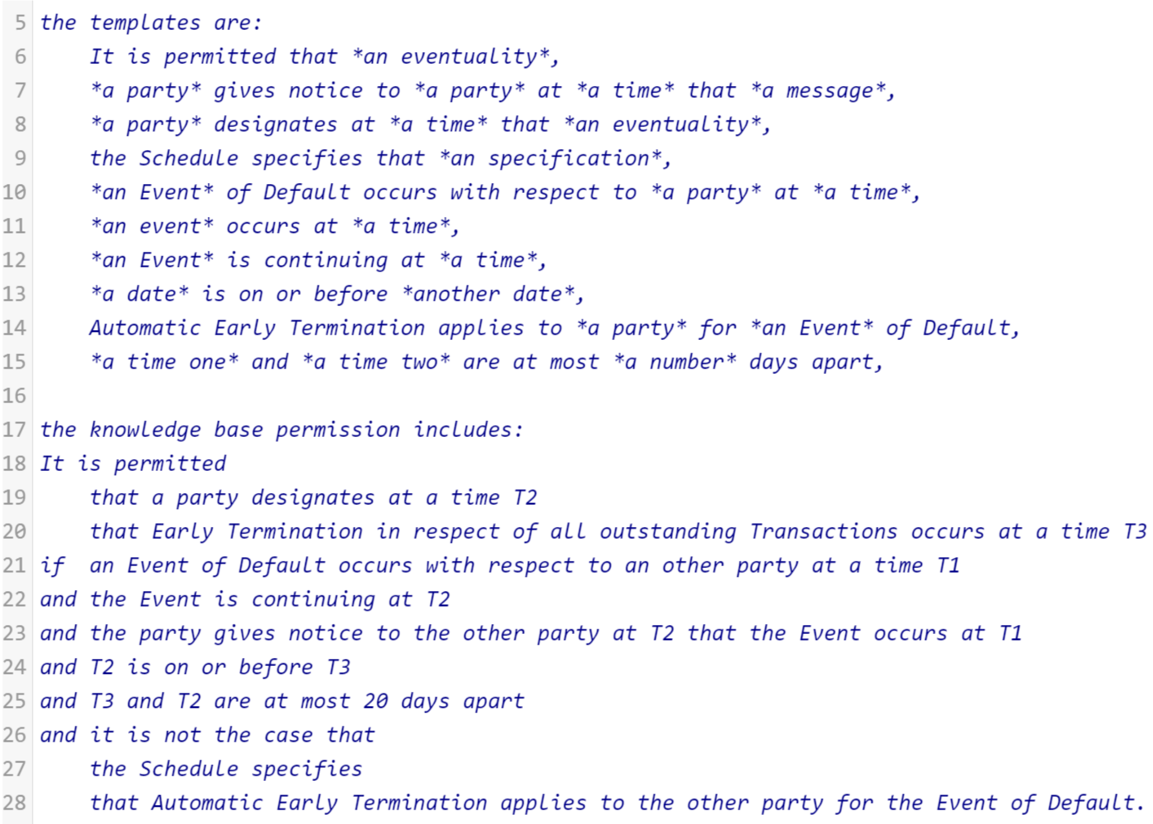
A meta-variable in a template is a parameter preceded by the keyword “that”. The keyword and parameter must be at the end of the template, followed by no other words.

A template containing a meta-variable is instantiated by replacing the meta-variable by a list of words representing a variable or by replacing it with another instance of a template. Templates containing meta-variables need to be declared in the list of templates before the templates that can be used to instantiate them.

For example:

[1] In formal logic, an instance of a predicate is an atomic formula containing free variables. However, in formal grammar, an instance of a predicate reads as a complete sentence with variables understood as universally or existentially quantified depending on the context.

(The example can be found at https://logicalenglish.logicalcontracts.com/p/permitted%20that.pl)



```
 5 the templates are:
 6     It is permitted that *an eventuality*,
 7     *a party* gives notice to *a party* at *a time* that *a message*,
 8     *a party* designates at *a time* that *an eventuality*,
 9     the Schedule specifies that *an specification*,
10     *an Event* of Default occurs with respect to *a party* at *a time*,
11     *an event* occurs at *a time*,
12     *an Event* is continuing at *a time*,
13     *a date* is on or before *another date*,
14     Automatic Early Termination applies to *a party* for *an Event* of Default,
15     *a time one* and *a time two* are at most *a number* days apart,
16 
17 the knowledge base permission includes:
18 It is permitted
19     that a party designates at a time T2
20     that Early Termination in respect of all outstanding Transactions occurs at a time T3
21 if  an Event of Default occurs with respect to an other party at a time T1
22 and the Event is continuing at T2
23 and the party gives notice to the other party at T2 that the Event occurs at T1
24 and T2 is on or before T3
25 and T3 and T2 are at most 20 days apart
26 and it is not the case that
27     the Schedule specifies
28     that Automatic Early Termination applies to the other party for the Event of Default.
```

Notice that the complex condition on line 18 contains one instance of a meta-variable inside another instance. Each condition must be written on a separate line, unless it is a condition with the word "that".

5. Knowledge Base and Scenarios

A knowledge base is a list of facts and rules separated by a new line. A scenario is also a list of facts and rules. separated by a new line.

There can be several knowledge bases in one document, intermixed with queries and scenarios. However, the current implementation uses all the facts and rules in all the knowledge bases (KBs) in the document to answer a query with a given scenario. We plan to exploit the ability to combine different KBs in the same document in the future. For example, to allow one KB to override another.

6. Facts and Rules

A fact is an instance of a template that ends with a full stop. Therefore, facts can contain variables.

In English it is natural to read such variables as existentially quantified, as in:

> a person is a British citizen on 2021-10-09.

However, the current implementation treats the variable “a person” as universally quantified, as in Prolog. This discrepancy will be addressed in future work. In some cases, however, the writer may want to actually use an universally quantified variable. For those, it is possible use the word "each" as a determiner to write, for example:

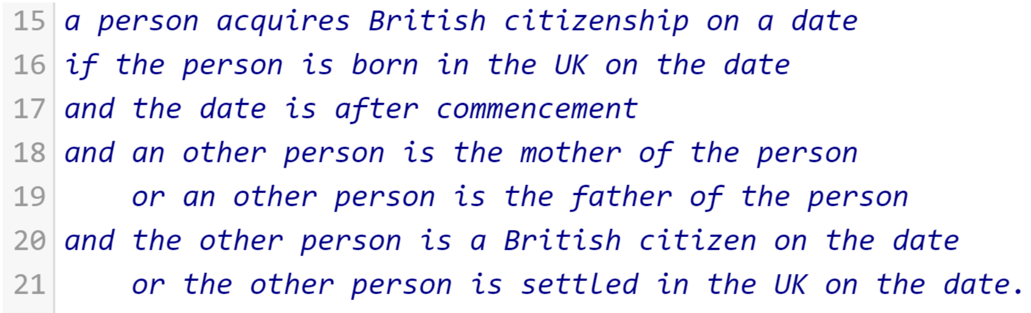
> each person is being considered.

A rule is a conclusion followed by the word "if", followed by one or more conditions separated by logical connectives "and" or "or", followed by a full stop.

A conclusion is an instance of a template.
A condition is a simple condition, which is an instance of a template,
or a complex condition.

Rules are sensitive to indentation. Indentation is defined as zero or more spaces between an end of line and the beginning of a condition.

Each condition (except the first condition, which can share a line with the conclusion of a rule) is allocated its own separate line, preceded by indentation and the corresponding logical connective. For example:



```
a person acquires British citizenship on a date
if the person is born in the UK on the date
and the date is after commencement
and an other person is the mother of the person
    or an other person is the father of the person
and the other person is a British citizen on the date
    or the other person is settled in the UK on the date.
```

Notice that the different logical connectives “and” and “or” must have different levels of indentation. Otherwise, their precedence would be indeterminate.

7. Complex Conditions

7.1 Negative conditions

The simplest complex condition is a negative condition, which is a list of the words:

> “it is not the case that”
> followed by a new line and an indentation, followed by one or more conditions separated by the logical connectives "and" or "or".

For example[2]:

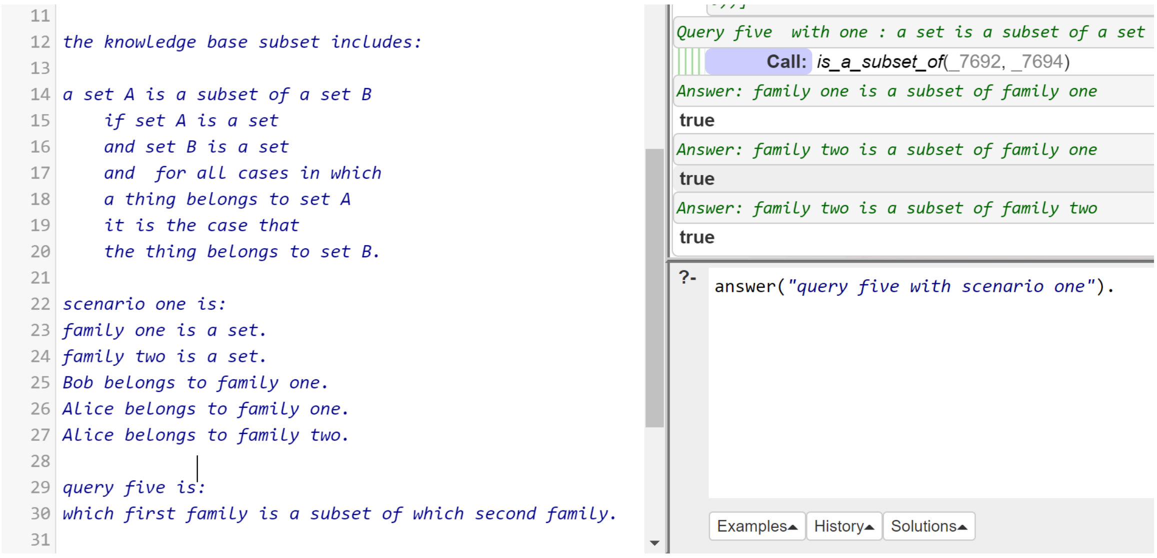
```
a person is born in the UK on a date
    if it is not the case that
    the person is born outside the UK on the date.
```

7.2 For-all condition

A for-all condition is a list of the words:

> “for all cases in which”
> followed by a new line and an indentation, followed by one or more conditions separated by the logical connectives "and" or "or",
> followed by a new” line and an indentation and the list of words
> “it is the case that”
> followed by a new line and an indentation, followed by one or more conditions separated by the logical connectives "and" or "or".

For example:



```
11
12 the knowledge base subset includes:
13
14 a set A is a subset of a set B
15     if set A is a set
16     and set B is a set
17     and  for all cases in which
18     a thing belongs to set A
19     it is the case that
20     the thing belongs to set B.
21
22 scenario one is:
23 family one is a set.
24 family two is a set.
25 Bob belongs to family one.
26 Alice belongs to family one.
27 Alice belongs to family two.
28
29 query five is:
30 which first family is a subset of which second family.
31
```

```
Query five  with one : a set is a subset of a set
Call: is_a_subset_of(_7692, _7694)
Answer: family one is a subset of family one
true
Answer: family two is a subset of family one
true
Answer: family two is a subset of family two
true
?- answer("query five with scenario one").
```

A for-all condition in LE translates into the forall predicate in Prolog, and consequently for-all has the same restriction as forall, that it cannot generate variable bindings. This explains the need for lines 15, 16, 23 and 24 above.

7.3 Lists

[2] Notice that the rule in this example can be regarded as shifting the burden of proof from showing that a person was born in the UK, to showing that the person was not born outside the UK.

A list in LE is a sequence of terms separated by commas and bracketed (by [and]). It is very close to the form in Prolog. For instance [1,2,3] is a list.

LE also supports that important resource of Prolog whereby one obtains the head of the list and the tail, such as in:

```
  a list is [a head|a tail].
```

Users that may be discouraged by this specialized notation, has access to another resource. A special predefined template is being provided to the same end:

```
  *a list*  has *a thing* as head before *a rest of the list*
```

Notice that "`a thing`" is an element of the list and "`a rest of the list`" refers to the tail of the list, even if it is empty.

7.4 Sets and Sums

LE admits a predefined template with the form:

```
  *a set* is a set of *a term*
              where *a list of conditions*
```

to allow for propositions related to sets. A set is concretely represented as a bracketed list of terms. For example, this is a possible instance:

```
a record of previous owners is a set of [an owner, a share]
       where the asset is ultimately owned by the owner with the
share at the previous time
```

Similarly, there is predefined templates to support addition of numbers in a list:

```
  *a number* is the sum of each *number* such that
       *a list of conditions*
```

to support aggregations such as in this instance:

```
  a number is the sum of each asset such that
       the asset is in [1000, 20, 50].
```

The word "every" can also be used instead of "each".

7.5 Built-in Predicates and operators

LE supports two kinds of built-ins, most of which are inherited from Prolog: built-in predicates and built-in operators.. Built-in predicates are represented by special templates, as shown in the following table 1:

Prolog	**LE predefined template**
append(A, B, C)	appending *an A* then *a B* gives *a C*
reverse(A, B)	*an A* is the reverse of *a B*
same_date(T1, T2)	*a T1* is the same date as *a T2*
between(Minimum,Maximum,Middle)	*a Middle* is between *a Minimum* & *a Maximum*
is_1_day_after(A, B)	*an A* is 1 day after *a B*
is_days_after(A, B, C)	*an A* is *a B* days after *a C*
immediately_before(T1, T2)	*a T1* is immediately before *a T2*
T1 \= T2	*a T1* is different from *a T2*
T1 == T2	*a T1* is equivalent to *a T2*
is_a(Object, Type)	*an Object* is of type *a Type*
is_not_before(T1, T2)	*a T1* is not before *a T2*
T1 = T2	*a T1* is equal to *a T2*
before(T1, T2)	*a T1* is before *a T2*
after(T1, T2)	*a T1* is after *a T2*
member(Member, List)	*a Member* is in *a List*
A is B	*a A* is *a B*
T1 \=@= T2	*a T1* \=@= *a T2*
T1 \== T2	*a T1* \== *a T2*
T1 =\= T2	*a T1* =\= *a T2*
T1 =@= T2	*a T1* =@= *a T2*
T1 == T2	*a T1* == *a T2*
T1 =< T2	*a T1* =< *a T2*
T1 >= T2	*a T1* >= *a T2*
T1 = T2	*a T1* = *a T2*

T1 < T2	*a T1* < *a T2*
T1 > T2	*a T1* > *a T2*
unparse_time(Number of Secs, Date)	*a Number of Secs* corresponds to date *a Date*

Table 1

Besides built-in predicates, Prolog built-in binary operators can be included in LE expressions with this structure:

```
  *a term* <a binary operator> *another term*.
```

7.6 Time and date predicates

As the reader may have noticed, there is some special support for times and dates. In particular, any expression that matches this structure:

a Year-*a Month*-*a Day*

or this

a Year-*a Month*-*a Day*T*a Hour*:*a Minute*:*a Second*

will be translated into the number of seconds since the 1rst of January 1970. That is since

1970-01-01T00:00:00

This is the ISO standard for dates. The format is particularly difficult in that every number represents an element of a date and, therefore, has to be written with the indicated number of digits and leading zeros if they are below 10. Otherwise, the expression will not be interpreted correctly.

Alternatively, LE writers are allowed to use simple numbers to refer to times (a number below 100 will not be translated into a number of seconds).

Templates such as

a T1 is before *a T2*,
a T1 is immediately before *a T2*,
a T1 is after *a T2*.

support those features.

Moreover, whenever a variable has been named with a set of words that includes the words "date" or "day" and the corresponding value is a number, the corresponding answer for a query will be shown in the standard format.

8. Queries

A query is a list of one or more query conditions separated by logical connectives "and" or "or", followed by a full stop.

A query condition is a condition in which the article introducing a variable may be replaced by the interrogative determiner “which”.

For example, the following queries are all equivalent:

> which first family is a subset of which second family.
> which first family is a subset of a second family.
> a first family is a subset of which second family.
> a first family is a subset of a second family.

However, these queries are different from the query:

> which family is a subset of which family.

which asks which family is a subset of itself.

9. Querying machinery

LE currently depends on the prolog querying facilities of the web platform SWISH. Queries are normally written in a special pane and then posted to the Prolog interpreter.

To submit a query, use the word "answer" followed by either a direct expression of the query written as described in the previous section or the identifier of the corresponding expressions declared as part of the document.

```
?- answer <query name>.
```

The answer predicate also allows for the identification of the scenario:

```
?- answer <query name> with <scenario name>.
```

For instance:

> ? answer one with test.

And for experience prolog users, with parentheses and quotes:

```
? answer('query one with scenario test').
? answer('one with test').
? answer('query one', with(test)).
? answer('query one', with(test), Answers).
? answer('query one', with(test), Explanations, Answers).
```

Notice that a direct expression of a query can be used:

```
? answer ' which thing followed by which other thing is [b, c]'
```

The LE command/query

```
?- show prolog.
```

shows the prolog translation of the whole LE document.

```
?- show rules.
```

shows the prolog translation of the rules in the LE document.

```
?- show queries
```

shows the prolog translation of the queries in the LE document.

```
?- show scenarios
```

shows the prolog translation of the scenarios in the LE document.

```
?-  show templates
```

shows the prolog translation of all the available LE templates (including predefined).

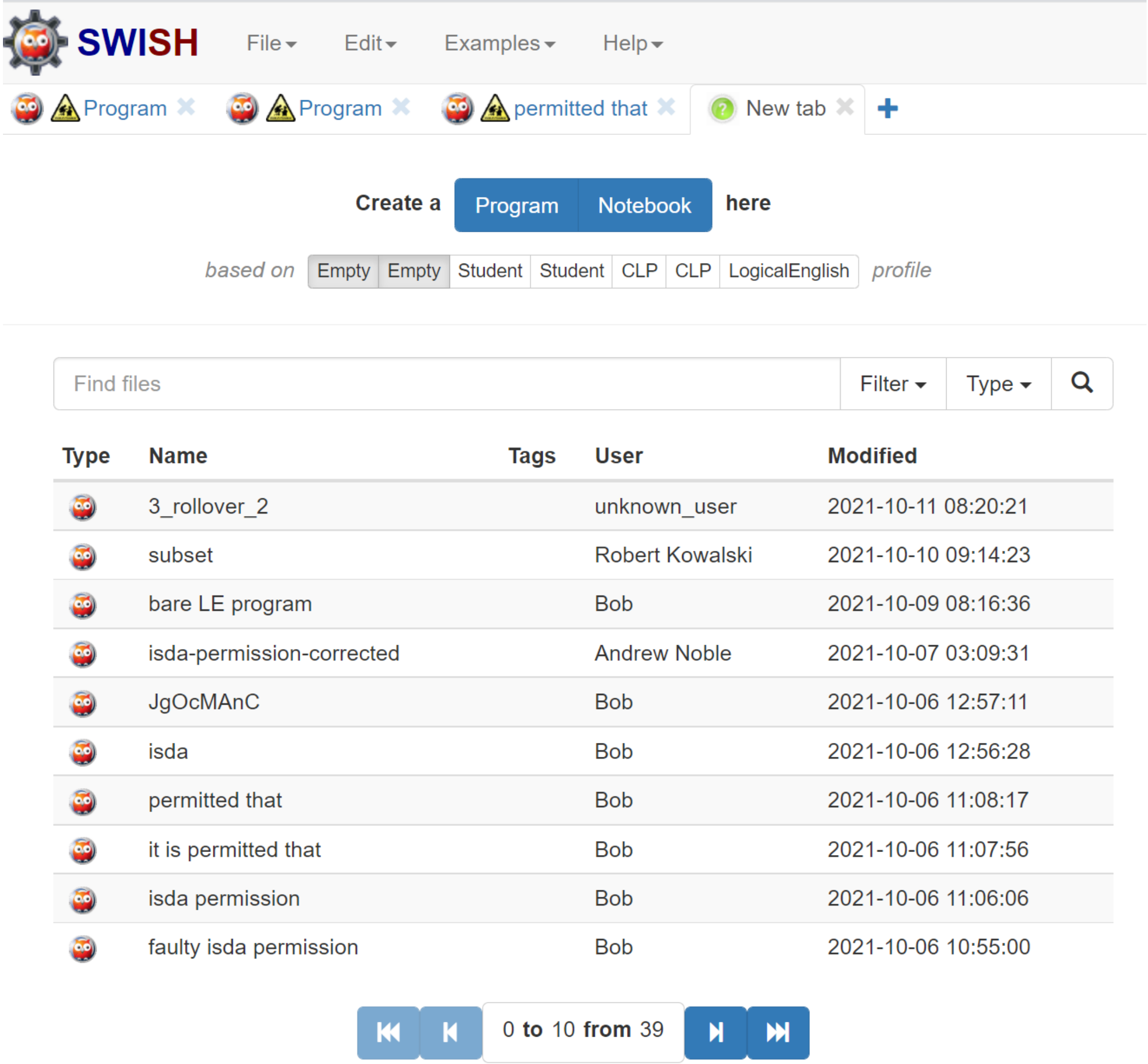
10. Other Features

The current implementation contains other features that need further testing, both to ensure that they work and to ensure that they are needed. These include:

- Complex conditions corresponding to aggregate operators in Prolog.
- List processing.
- Event calculus with event templates, fluent templates and metatemplates for defining meta-predicates with more than one predicate argument (such as the predicates for initiating and terminating fluents).

11. Example programs

Several Prolog programs can be open in the left pane of a SWISH screen, some of which may contain LE documents. To create a new program or document, click on the plus symbol +. This will open a list of previously saved Prolog programs, including LE documents. Most of these documents are out of date, and were used for debugging the interpreter. The most recent programs are more likely to work than the older ones.



Type	Name	Tags	User	Modified
	3_rollover_2		unknown_user	2021-10-11 08:20:21
	subset		Robert Kowalski	2021-10-10 09:14:23
	bare LE program		Bob	2021-10-09 08:16:36
	isda-permission-corrected		Andrew Noble	2021-10-07 03:09:31
	JgOcMAnC		Bob	2021-10-06 12:57:11
	isda		Bob	2021-10-06 12:56:28
	permitted that		Bob	2021-10-06 11:08:17
	it is permitted that		Bob	2021-10-06 11:07:56
	isda permission		Bob	2021-10-06 11:06:06
	faulty isda permission		Bob	2021-10-06 10:55:00