Artificial Intelligence with Erlang: the Domain of Relatives

From Erlang Community

Contents

Introduction

This article describe how to write a simple Artificial Intelligence application with Erlang, by using a *rule production system*.

To reach this objective, we will exploit the **ERESYE** tool, an Erlang Inference Engine that allows rule-based systems to be written directly in Erlang.

As it is widely known, a rule-based system is composed by a **knowledge base**, which stores a set of *facts* representing the "universe of discourse" of a given application, and a set of **production rules**, which are used to infer knowledge and/or reason about the knowledge. A rule is activated when one or more facts match the template(s) given in the rule declaration: in such a case, the body of the rule contains a code that is thus executed

In ERESYE, *facts* are expressed by means of Erlang tuples or records, while rules are written using standard Erlang function clauses, whose declaration reports, in the clause head, the facts or fact templates that have to be matched for the rule to be activated and executed.

For more information about ERESYE please refer to the paper:
A. Di Stefano, F. Gangemi, and C. Santoro. ***ERESYE: Artificial Intelligence in Erlang Programs.*** *In ERLANG'05: Proceedings of the 2005 ACM SIGPLAN workshop on Erlang, pages 62-71, New York, NY, USA, 2005. ACM Press.*
You can also take a look at the slides
(http://www.erlang-consulting.com/erlangworkshop05/eresye.pdf).

For more information about rule-based inference engines and expert systems, you can refer to the book:
S. Russell and P. Norvig. ***Artificial Intelligence: A Modern Approach/2E.*** *Prentice Hall, 2003.*

To write an AI application with ERESYE the following steps have to be performed:

1. Indentify your universe of discourse and determine the facts that have to be used to represent such a world;
2. Indentify the rules that you need and write them by using, e.g. first-order-logic predicates or even natural language;
3. Implement the system by writing your rules as Erlang function clauses, according to the modality required by ERESYE.

Surely you have to obtain the ERESYE tool: it is not publicly advertised on a web page but it can be easily obtained by dropping an email to one of its author, Francesca Gangemi (francesca@erlang-consulting.com) or Corrado Santoro (csanto@diit.unict.it).

Please note that this example works only with the latest release of ERESYE, which is 1.2.4 (contact authors for it). Old releases may cause problems.

The Application: the Domain of Relatives

We will design a system able to derive new knowledge using some inference rules and starting from a small set; as a sample application, we chose the domain of relatives: we will start from some base concepts, such as *parent*, *male* and *female*, and then, by means of a proper set of rules, we will derive the concepts of *mother*, *father*, *sister*, *brother*, *grandmother* and *grandfather*.

According to the list above, we will first derive the facts that will be used to represent our concepts. Given the set of relationships above, they will be represented by means of the following facts:

#	Concept	Fact / Erlang tuple
1	X is male	`{male, X}`
2	X is female	`{female, X}`
3	X is Y’s parent	`{parent, X, Y}`
4	X is Y’s mother	`{mother, X, Y}`
5	X is Y’s father	`{father, X, Y}`
6	X is Y’s sister	`{sister, X, Y}`
7	X is Y’s brother	`{brother, X, Y}`
8	X is Y’s grandmother	`{grandmother, X, Y}`
9	X is Y’s grandfather	`{grandfather, X, Y}`

Concepts 1, 2 and 3 will be used as a base to derive the other ones.

Deriving new concepts by means of rules

Concept: "mother". The rule to derive the concept of mother is quite straightforward: *if X is female and X is Y’s parent then X is Y’s mother.*

From the point of view of ERESYE, since knowledge is stored in the *knowledge base* of the engine, the rule above is translated into the following one: *if the facts `{female, X}` and `{parent, X, Y}` are **asserted** in the knowledge base, then we assert the fact `{mother, X, Y}`.*

The rule `mother` can be thus written as follows:

```
%%
%% if (X is female) and (X is Y’s parent) then (X is Y’s mother)
%%
mother (Engine, {female, X}, {parent, X, Y}) ->
  eresye:assert (Engine, {mother, X, Y}).
```

Concept: "father". This concept can be easily derived by means of the following rule:

```
%%
%% if (X is male) and (X is Y’s parent) then (X is Y’s father)
%%
father (Engine, {male, X}, {parent, X, Y}) ->
  eresye:assert (Engine, {father, X, Y}).
```

Concept: "sister". This concept can be expressed by the following rule: *if Y and Z have the same parent and Z is female, then Z is the Y’s sister.* The ERESYE rule used to map this concept is:

```
%%
%% if (Y and Z have the same parent X) and (Z is female)
%%    then (Z is Y’s sister)
%%
sister (Engine, {parent, X, Y}, {parent, X, Z}, {female, Z}) when Y =/= Z ->
  eresye:assert (Engine, {sister, Z, Y}).
```

Please note the guard, which is needed to ensure that when Y and Z are bound to the same value, the rule is not activated (indeed this is possible since the same fact can match both the first and second "parent" pattern).

Concept: "brother". Given the previous one, this concept is now quite simple to implement:

```
%%
%% if (Y and Z have the same parent X) and (Z is male)
%%    then (Z is Y’s brother)
%%
brother (Engine, {parent, X, Y}, {parent, X, Z}, {male, Z}) when Y =/= Z ->
  eresye:assert (Engine, {brother, Z, Y}).
```

Concepts: "grandmother" and "grandfather". The former concept can be expressed by means of the rule: *if X is Y’s mother and Y is Z’s parent, then X is Z’s grandmother.* The latter concept is now obvious. Both can be implemented using the following ERESYE rules:

```
%%
%% if (X is Y’s mother) and (Y is Z’s parent)
%%    then (X is Z’s grandmother)
%%
grandmother (Engine, {mother, X, Y}, {parent, Y, Z}) ->
  eresye:assert (Engine, {grandmother, X, Z}).
```

```
%%
%% if (X is Y’s father) and (Y is Z’s parent)
%%    then (X is Z’s grandfather)
%%
grandfather (Engine, {father, X, Y}, {parent, Y, Z}) ->
  eresye:assert (Engine, {grandfather, X, Z}).
```

Instantiating the Engine and Populating the Knowledge Base

After writing the rules, we need to:

1. instantiate the engine;
2. add the rules above to the engine;
3. populate the knowledge base with a set of initial facts.

We do this in the function `start` below:

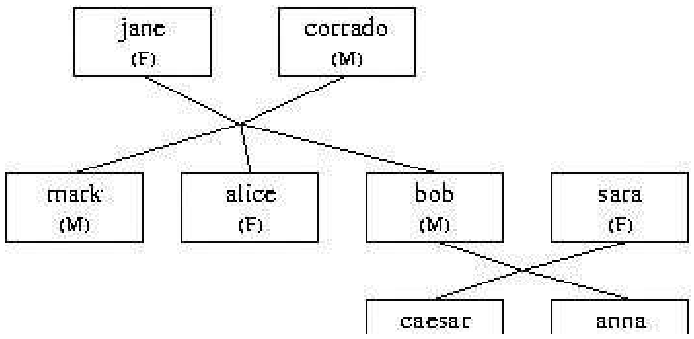
```
start () ->
  eresye:start (relatives),
  lists:foreach (fun (X) ->
                     eresye:add_rule (relatives, {?MODULE, X})
                 end,
                 [mother, father, brother, sister,
                  grandfather, grandmother]),

  eresye:assert (relatives,
                 [{male, bob}, {male, corrado}, {male, mark}, {male, caesar},
                  {female, alice}, {female, sara}, {female, jane}, {female, anna},
                  {parent, jane, bob}, {parent, corrado, bob},
                  {parent, jane, mark}, {parent, corrado, mark},
                  {parent, jane, alice}, {parent, corrado, alice},
                  {parent, bob, caesar}, {parent, bob, anna},
                  {parent, sara, casear}, {parent, sara, anna}]),
  ok.
```

As the listing reports, creating a new ERESYE engine implies to call the function `eresye:start/1`, giving the name of the engine to be created.

Then, we have to add the rules to the engine by using the function `eresye:add_rule/2`: it takes two arguments, the name of the engine and a tuple representing the function in the form `{Module, FuncName}`; obviously the function `Module:FuncName` must be exported. Function `add_rule` has to be called for each rule that has to be added; for this reason, the code above has an iteration over the list of rules written before.

Finally, we populate the inference engine with a set of sample facts by giving them, in a list, to the function `eresye:assert/2`. To test our rules, we considered the relationships in the Figure below and assert only the facts for *male*, *female* and *parent*.



Testing the application

The final complete code of our AI application is thus the following:

```
%
% relatives.erl
%
-module (relatives).
-compile ([export_all]).

%%
%% if (X is female) and (X is Y’s parent) then (X is Y’s mother)
%%
mother (Engine, {female, X}, {parent, X, Y}) ->
  eresye:assert (Engine, {mother, X, Y}).


%%
%% if (X is male) and (X is Y’s parent) then (X is Y’s father)
%%
father (Engine, {male, X}, {parent, X, Y}) ->
  eresye:assert (Engine, {father, X, Y}).



%%
%% if (Y and Z have the same parent X) and (Z is female)
%%    then (Z is Y’s sister)
%%
sister (Engine, {parent, X, Y}, {parent, X, Z}, {female, Z}) when Y =/= Z ->
  eresye:assert (Engine, {sister, Z, Y}).



%%
%% if (Y and Z have the same parent X) and (Z is male)
%%    then (Z is Y’s brother)
%%
brother (Engine, {parent, X, Y}, {parent, X, Z}, {male, Z}) when Y =/= Z ->
  eresye:assert (Engine, {brother, Z, Y}).


%%
%% if (X is Y’s father) and (Y is Z’s parent)
%%    then (X is Z’s grandfather)
%%
grandfather (Engine, {father, X, Y}, {parent, Y, Z}) ->
  eresye:assert (Engine, {grandfather, X, Z}).


%%
%% if (X is Y’s mother) and (Y is Z’s parent)
%%    then (X is Z’s grandmother)
%%
grandmother (Engine, {mother, X, Y}, {parent, Y, Z}) ->
  eresye:assert (Engine, {grandmother, X, Z}).



start () ->
  eresye:start (relatives),
  lists:foreach (fun (X) ->
                     eresye:add_rule (relatives, {?MODULE, X})
                 end,
                 [mother, father,
                  brother, sister,
                  grandfather, grandmother]),

  eresye:assert (relatives,
                 [{male, bob},
                  {male, corrado},
                  {male, mark},
                  {male, caesar},
                  {female, alice},
                  {female, sara},
                  {female, jane},
                  {female, anna},
                  {parent, jane, bob},
                  {parent, corrado, bob},
                  {parent, jane, mark},
                  {parent, corrado, mark},
                  {parent, jane, alice},
                  {parent, corrado, alice},
                  {parent, bob, caesar},
                  {parent, bob, anna},
                  {parent, sara, casear},
                  {parent, sara, anna}]),
  ok.
```

Now it’s time to test our application:

```
Erlang (BEAM) emulator version 5.5 [source] [async-threads:0] [hipe]

Eshell V5.5  (abort with ^G)
1> c(relatives).
{ok,relatives}
2> relatives:start().
ok
3>
```

Following the call to function `relatives:start/0`, the engine is created and populated; if no errors occurred, the rules should have been processed and the new facts derived. To check this, we can use the function `eresye:get_kb/1`, which returns the list of facts asserted into the knowledge base of a given engine:

```
4> eresye:get_kb(relatives).
[{brother,bob,mark},
 {sister,alice,bob},
 {sister,alice,mark},
 {brother,bob,alice},
 {brother,mark,alice},
 {grandmother,jane,caesar},
 {grandfather,corrado,caesar},
 {grandmother,jane,anna},
 {grandfather,corrado,anna},
 {sister,anna,caesar},
 {brother,caesar,anna},
 {sister,anna,casear},
 {mother,sara,anna},
 {mother,sara,casear},
 {parent,sara,anna},
 {father,bob,anna},
 {parent,sara,casear},
 {father,bob,caesar},
 {parent,bob,anna},
 {father,corrado,alice},
 {parent,bob,caesar},
 {mother,jane,alice},
 {parent,corrado,alice},
 {father,corrado,mark},
 {parent,jane,alice},
 {mother,jane,mark},
 {parent,corrado|...},
 {brother|...},
 {...}|...]
5>
```

The presence of facts representing concepts like *father*, *sister*, etc., proves that the rules seems to be working as expected.

We can however query the knowledge base using specific fact templates. For example, if we want to know who are Alice’s brothers, we can use the function `eresye:query_kb/2` as follows:

```
6> eresye:query_kb(relatives, {brother, '_', alice}).
[{brother,bob,alice},{brother,mark,alice}]
7>
```

The facts returned conform to the relationships depicted in the figure above, thus proving that the rules written are really working.

As the example shows, function `eresye:query_kb/2` takes the engine name as the first argument, while, for the second parameter, a tuple has to be specified, representing the fact template to be matched; in such a tuple, the atom `'_'` plays the role of a wildcard. However, to specify a more complex matching, a `fun` can be used as a tuple element; this `fun` has to return a boolean value which indicates if the element matches the template. For example, to select both Alice's and Anna's brothers, we can use the following function call:

```
7> eresye:query_kb(relatives, {brother, '_', fun (X) -> (X == alice) or (X == anna) end}).
[{brother,bob,alice},{brother,mark,alice},{brother,caesar,anna}]
8>
```

Conclusions

This HowTo not only shows how to use the ERESYE engine to write an AI application, but also highlights the versatility of the Erlang language: the characteristics of functional and symbolic programming, together with the possibility of performing *introspection* of function declaration, can be successfully exploited for application domains which are completely new for Erlang but can surely be very interesting.

Retrieved from "http://www.trapexit.org/index.php/Artificial_Intelligence_with_Erlang:_the_Domain_of_Relatives"

Categories: HowTo | AI

Hosted by Erlang Training and Consulting