Theo representational semantics and conventions.

Tom Mitchell
September 2009

This document describes key details of the Theo knowledge representation.

Special status of "Specializations" and "Generalizations" slots

Most slots in Theo are defined by the user. A few dozen come with Theo, and are part of the definition of Theo itself (which can be changed by modifying these). Two of the Theo-provided slots, "specializations" and "generalizations" have special status because they define the inheritance hierarchy of entities. They are special in the following sense:

1. if you addValue(generalizations, x, y), then Theo will also automatically addValue(specializations, y, x). Similarly, if you addValue(specializations, y, x), Theo will automatically addValue(generalizations, x, y). In short, Theo maintains the consistency of the generalizations and specializations slots. This enables it to always efficiently find the transitive closure of both relations, which is very useful for operations like inheritance.

2. The specializations of x must not contain y if it also contains a generalization of y. Similarly, generalizations of y must not contain x if it also contains a specialization of x. This is not currently enforced by the Theo addValue function, but you should make sure your own code enforces it, because other Theo code (e.g., for inheritance) depends on this assumption.

If you create or edit your KB, be sure to follow the above two conditions.

Number of values a slot can have.

Different slots can have different numbers of values. The slot nrOfValues denotes the number allowed for any given slot. For example, everybody has only one mother, which we assert in Theo by the assertion nrOfValues(mother)=1. However, we have two parents, hence nrOfValues(parents)=2, and any number of daughters, hence nrOfValues(daughters)='any'. The legal values for nrOfValues are any integer, or the string 'any'.

Representing slots with no values.

Slots can have no value. There are two different situations in which this can occur: (1) the value of the slot is not known, hence it has no *known* value, or (2) the slot is known to have a null or empty value. For example, the generalizations(everything)={}, because 'everything' is the root of all entities in

Theo, and it literally has no generalizations. This is case 2. In contrast, the age(BillGates) might be unknown, but he clearly does have an age. This is case 1.

Theo represents the value in the first case as NO_THEO_VALUE. It represents the value in the second case as the empty list {}. (this is the case regardless of the nrOfValues for the slot). If you query getValue(slot, entity), and slot is not even present in the data structure that defines entity, then getValue will return NO_THEO_VALUE.

Storing sources of slot values.

Slot values can optionally have sources, or justifications, in Theo. In the RTW system the sources for the value of slot s of entity e are stored as a list in the "source" subslot (that is, in the "source" slot of {e s}). If slot s contains k values, then its source subslot will also contain k values, where the ith value of source justifies the ith value of s. Each item in the list of sources (we'll call it a source) is itself a list, to allow for the possibility that several different methods have proposed this value. Each item in this list (we'll call it a source item) is a list whose format is not constrained, but we suggest the following form for each source item:

{<methodName> <argumentList> <anyOtherRelevantInformation>}

such as the following justification used by the "prolog" method, which contains sufficient information to recalculate the slot value which is justified by this source. The first item here is the name of the method, and the second item is the information needed by the prolog method: first the rule, then the list of its variables, then the items that bound to these variables to infer the belief that the "company_economic_sector" of the company "excite" is "media."

```
{prolog,
 {{{{company_economic_sector, ?x, ?y} , {competes_with, ?x, ?z} ,
    {company_economic_sector, ?z, ?y} } , 0.7761, 131, 33, 305} ,
  {?x, ?z, ?y} ,
  {excite, lycos, media} } } }
```

Upper/lower case issues

In principle, you can define a Theo entity named "OranGe" and another named "orange" and inside core memory, these will be kept separate. However, it seems the .xml files that represent that KB on disk are all lower-case filenames (ask Andy and Tom why). Therefore, it is strongly discouraged to have two distinct entity names that differ only by upper/lower case. Just don't do it.

Swapping the KB from disk in Matlab implementation of Theo.

When using very large KB's (e.g., with 50K or more entities) it can be useful to store the KB on disk, and have Theo work directly from the disk copy. To do this, set global variables as follows

```
THEO.kbdir='/Users/tommitchell/kb/';  % root directory of the KB
THEO.readDiskKB=1; % enable loading KB entities on demand from disk
THEO.maintainDiskKB=1; % enable writing out KB updates to disk
THEO.maxEntitiesInRAM=10000; % number of KB entities to keep cached in RAM
THEO.traceSwapInEntity=1; % enable screen notification when entities swapped in
```

The primary Theo API functions to handle disk swapping are

```
useKB(kbdirectory); % sets THEO.kbdir to its input argument
isEntityInRAM(entity); % returns 1 if entity is cached in RAM, else 0
isEntityOnDisk(entity); % Returns 1 if entity exists on THEO.kbdir, else 0
swapInEntity(entity, <dir THEO.kbdir>) % swaps in entity from disk to RAM
swapOutEntity(entity, <saveOnDisk 0>) % swaps out entity, optionally saving first to disk.
```