Context Errors

Richard H. McCullough

Pioneer, CA 95666
rhm@cdepot.net

Abstract

The definition of context is reviewed, and context errors which occur in the real world are discussed. Common types of errors are identified, along with general methods of correcting these errors. Several real world examples are presented.

1. Context

Context is the name of a proposition list

Every proposition has a context, which is a list of other propositions. The "view" attribute names that proposition list. A context may have "space", "time" subcontexts which specify where, when actions occur.

Genus-differentia definitions

The goal to strive for: a genus-differentia definition for every concept.

ECP hierarchy (knit)

A context is represented as an entity-characteristic-proposition (ECP) hierarchy, which is the fundamental knowledge unit (knit).

Actions

The goal to strive for: fully characterize all actions, without suppressing any important information.

The MKR language

All of the above principles have been implemented in the MKR language. MKR is a user-friendly RI ("Real Intelligence") language. MKR uses terse English-like propositions to help a human user focus on the essentials, and avoid floating abstractions.

McCullough Knowledge Explorer (MKE)

as The MKE program is the core of a knowledge base (KB) system which uses the MKR language as a user interface. MKE includes an intelligent knowledge assistant (ke) which helps the user record/change/search knowledge, checks the knowledge for inconsistencies, and executes user commands.

The rest of this paper discusses the kinds of errors that occur and how those errors are corrected, and gives several real world examples.

2. Error classes

Unknown concepts

MKE classifies every concept in its input stream. If the user doesn't specify a genus, MKE classifies the concept as unknown.

Undefined concepts

MKE checks every concept to see if a genus-differentia definition has been specified. If not, the concept is undefined.

Ambiguous concepts

MKE checks the genus of each concept. If it has more than one genus, it is ambiguous.

Individual-concept conflicts

If a concept (which is an abstraction) has also been specified to be an individual (which is a concrete thing), MKE flags it as an error.

Loops

Since loops in the ECP hierarchy can cause infinite loops in inference programs, MKE flags any loops as errors.

Broken links

The internal structure of the ECP hierarchy uses many double-linked lists (for faster KB processing). Some MKE "bugs" will delete one of these links. MKE checks itself by flagging broken links as errors.

3. Correcting errors

Changing names

Ambiguous concepts usually indicate that two distinct concepts were inadvertently given the same name. The solution is easy – give each distinct concept a distinct name.

Classifying unknowns

Sometimes an unknown concept is simply the result of "reference before definition". The MKE command

```
do check od unknown done;
```

checks for this case (concept has one other genus besides unknown), and reclassifies the concept. If the concept has more than one other genus, then it is ambiguous.

Integration and differentiation

If the user notices that a concept has the wrong genus

```
concept  iss wrong genus;
```

it can be corrected by adding this statement to the KB

```
concept isd right genus;
```

If a new concept needs to be inserted into the KB, it can be done with

```
new concept isi species1, species2, … ;
```

Simplifying the lattice

Ambiguous concepts may arise from merging several "partial" contexts. Often, the ambiguity can be resolved by keeping only the most specific reference to the concept. For example, the hierarchy

```
car
/   Bessie
/   Chevy
//      Bessie
```

can be replaced by

```
car
/   Chevy
//      Bessie
```

The MKE command

```
do simplify lattice od car done;
```

automatically makes any changes of this type in the car subhierarchy.

4. Some examples

man and woman

In the MKR knowledge base, the first use of "man" is in the broad philosophical sense, as used by Aristotle.

```
man iss animal with rational;
```

which corresponds to the hierarchy

```
entity
/   animal
//      man
```

This differs from the every day use of animal and man, which
Corresponds to the hierarchy

```
entity
/   animal
/   person
//      man
//      woman
```

Merging these two contexts, we get the hierarchy

```
Entity
/   animal
//      man
/   person
//      man
//      woman
```

MKE automatically detected the ambiguity in man. We can "correct" man by differentiation.

```
man isd person;
```

set

In the early version of MKE, set had two different meanings: the verb which denotes assignment, and the mathematical group.

```
set charformat = column;
```

set iss group with inclusive, no order;

MKE automatically detected this ambiguity, and I decided to use set and Set for these two concepts. More recently, I decided that I would rather use the conventional lower case name for the mathematical group, so I changed the names to let and set.

urel, brel, trel, nrel

These are MKR verbs which denote unary, binary, ternary, n-ary relations. When I began to work with the OWL language, I needed to distinguish the different types of OWL properties in order to translate between OWL and MKR. I assigned a chartype attribute to each OWL property. For example

rdfs:subClassOf has chartype = brel;

MKE immediately rejected this statement – it's a syntax error, because brel is a verb.
So I changed my description to

Rdfs:subClassOf has chartype = relation, arity=2;

owl:Class

The current definition of owl:Class is based on non-well-founded set theory. When MKE processes an OWL ontology, it detects two types of errors: infinite loops and "strange things" which are both individuals and classes. For all of my work with OWL, I replace owl:Class (the class of all Classes) with owl:ClassSet (the set of all Classes), which produces a language governed by well-founded set theory.

5. Context interaction

Knowledge explorer (ke)

I often volunteer to help users who are trying to "debug" their ontologies. MKE is very helpful in these situations. It provides many error checks, useful KB statistics, and "pretty-prints" the ECP hierarchy with a simple command

Thing isc* ?;

Sometimes, MKE "confuses" the user, because the entities and properties of ke (and its ECP hierarchy) are merged together with the entities and properties of the user's ontology. Some of this confusion must be tolerated. If you delete the properties of ke, you have deleted its self-knowledge, and it is no longer an "intelligent" assistant.

Hiding a context

There is one simple "trick" which I have used to ease the confusion. I mark the ke properties as "hidden", so that they are not displayed when the user's hierarchy is displayed.

References

McCullough, R. H. 2006. *The MKR Language,*Forthcoming.