

CS246 - Answers / Grading Notes for Exercise Set 1

Exercise #1 (15 points)

5 points – Removing Thing as rootclass and making Book and Computer to be rootclasses.

10 points – Properly transforming the book years to two digits.

Exercise #1.1 (10 points)

In what way do tables and classes differ?

A class is a set of objects which share a common set of attributes. A table is a set of tuples. Classes can be organized into a class hierarchy, which allows one to provide hierarchical groupings among objects.

Notes: Some of you also mentioned that classes and relations can represent the same information. This is not quite true. Classes + attributes can represent the same information as any table and vice-versa, but classes alone cannot do this. Also, a class is not a relation. This is why we need predicates to tell us what is in a class and what is not.

Exercise #2 (40 points)

5 points – predicates created correctly

Notes: The predicates should have corresponding PredicateRelations with the proper domain (i.e. the proper domain of Person.Instance is Person). Some of you had predicates that were Relations but not PredicateRelations, which of course did not give you the option of filling in the proper domain.

15 points – attributes created correctly

Notes: The attributes should have corresponding AttributeRelations with the following domains and ranges. Also the attributes that are indisputably unique and/or total are marked as such:

| Attribute | Domain | Range | Total? | Unique? |
|---------------------|---------------|--------------|---------------|----------------|
| Person.Name | Person | String | yes | yes |
| Advisor.Student | Advisor | Student | | |
| Advisor.Institution | Advisor | Institution | | |
| Student.Advisor | Student | Advisor | | |
| Institution.Name | Institution | String | yes | yes |

Some of you were crafty enough to make Person.Name an attribute of Advisor and Student, which solved the inheritance problem. I gave bonus points for this.

5 points – class structure correct

Notes: Thing should be a superclass of Person and Institution. Also, of course Person should be the superclass of Advisor and Student.

5 points – data instances created correctly

Notes: Advisor.Student should be the mirror image of Student.Advisor. Also, the attributes that are objects and not strings should be handles of the appropriate type. I.e. Advisor.Student should be a student handle, not a string or an advisor handle. Also Students should be Persons, and Advisors should be Persons. Technically, all objects should also be created as Things, but I did not take off for this.

10 points – user interface works

Notes: For the search page, you need to make your string attributes be displayed as Stringfields, or the search won't work. (No points taken off for this, but you should know this for future assignments). Also, for your create and change pages, you should have drop-down lists for attributes with ranges that are objects. For object-ranged attributes, your search page should have a drop-down list for ease of searching, as well. You should have filled in values for the UI attributes Createstyle, Comparestyle, Inspectstyle, Changestyle and SearchStyle.

Exercise #2.1 (10 points)

The fundamental difference is that all of the objects referenced in SXML have handles. Also in SXML the relationships (attributes) are represented explicitly. In the Ullman tree, there are no handles or explicit relationships, which is why Infomaster does not have enough information to parse it out into individual facts. The relationships in the Ullman tree are implicitly represented by nesting. The handles and explicit relationships are what makes SXML “semantic.”

Notes: Some of you mentioned that handles allow one to reference the same object in multiple places, which is certainly a benefit of handles and SXML. Also, some of you noticed that the Ullman format might necessitate having duplicate data because of its structure. These are correct observations, but were over and above what we were looking for. Others of you focused on the syntactic structure of the documents, such as what is an attribute and what is an element, or how one is nested and the other is flat. Such syntactic considerations are unimportant. As mentioned in class, there are many equivalent syntactic variants of SXML, including nested/un-nested variants and attribute-based and element-based variants.

Exercise #3 (20 points)

You were graded on Prof. Genesereth's general impression of your work. Sorry but I don't have written comments on your presentations, just a score.

Exercise #4.1 (10 points)

KIF to TDC: We cannot always convert from KIF to TDC, because KIF can express logical information, while TDC cannot. For example: not, one of.

TDC to XML: We can easily make this conversion. Just add angle brackets!

XML to KIF: We can always remove the angle brackets and make them parentheses (un-nesting is also required). Example:

```
<person name="Mike">  
  <haircolor>brown</haircolor>  
</person>
```

can be represented in KIF as:

```
(person person1)  
(name person1 Mike)  
(haircolor haircolor1)  
(element person1 haircolor1)  
(contents haircolor1 brown)
```

Thus, KIF can express the same information as any XML document.

Notes: Some of you said that you cannot convert from XML to KIF unless it is SXML. While exercise 2.1 hinted that in general XML is not "semantic," we can always represent it in KIF. For example we can represent the sentence "This sentence is not semantic." in KIF as the following:

```
(sentence "This sentence is not semantic.")
```

Thus we see that the lack of semantics is a property the data itself, not just the data format.

Exercise #4.2 (5 points)

Here I was trying to get at the fact that if all of the links in the system touch the same node, then no index will be able to speed up the queries, and in fact will just slow down the system during update. Steven, Jiangjiang and Jing state this quite nicely:

“Our indexing method allows us to quickly get the list of links for a particular index value. We can index by either nodes or relations. So, to make the index useless and force the system to do a linear search through all the links, we’d want to have all the links attached to a single node, and have them all be the same relation type. To make indexing work well and speed the system up, we’d want to do just the opposite – distribute the links evenly across all nodes and relations.”

Notes: Apparently this question was ambiguous and people’s answers varied quite a bit. For example, some groups talked about queries that would run slow on Infomaster. (disconnected query graphs do the trick). Others talked about a good distribution of data in an integrated system (if the data is too distributed this can slow down the information retrieval). I gave credit to those answers that were correct, even if they weren’t what I originally intended.

Exercise #4.3 (5 points)

For dataweb data, KIF and OOKIF can represent the same information. However the syntax is a bit different, with OOKIF grouping by object (similar to SXML in a sense), and indenting nicely. I prefer OOKIF, since it is easier to read!

Notes: Every group preferred OOKIF to KIF for ease of reading, although some groups remarked that KIF is faster to load since it matches well with the internal indices in Infomaster.

Exercise #4.4 (5 points)

When deleting binary links from the dataweb, a compound index can speed things up, because we can look up the deleted link in the compound index. Similarly, when inserting a binary link in the dataweb, a compound index can speed things up when we do an integrity check to make sure that the link does not already exist in the dataweb.

When adding and deleting nodes and unary links from the dataweb, a compound index will only slow down the system, since the index needs to be updated but it cannot be used to speed up the insertion/deletion.

When searching, a compound index is not of much use. Note that in all “normal” query graphs, the leaf nodes of the graph are labeled by constants, and the internal nodes are labeled by variables (see lecture 3 for several examples of this). Then as there are no constants in adjacent nodes in the query graph, a compound index cannot be used.

Notes: One of the groups mentioned that a compound index would not be as useful if there were null values allowed in the database, which is true for the same reason that a compound index is not as useful when there are variable labels in the query graph. Also,

note that it is possible to have query graphs that contain adjacent nodes with constant labels. However, these graphs are not particularly useful and are very uncommon in practice (think about what adjacent constant nodes mean in a query!).