

TDDD43 - Lab Exercises for Theme NoSQL

Notice: Please make sure you have read the whole lab compendium before you start to work on the server from NSC.

Description and Aim

In this lab you will work on the Tetralith¹ set up which is the largest HPC cluster from the National Supercomputer Centre (NSC). You are supposed to solve an XPath query answering problem using the Hadoop system on Tetralith, by first generating the index structure of the given XML data (section 1.1) and then writing an XPath query in Java following Apache Hadoop 2.7.0² (section 1.2). Finally you need to compile your code and run the program on Tetralith (section 2). After completing this lab, you should have basic knowledge on the programming environment and the programming techniques of Hadoop.

1. XPath query answering

The task consists of two parts. In the first part, you will learn how to parse the XML file. In the second part, you will learn how to write a Java program using MapReduce based on Hadoop.

1.1 Parsing the XML file and storing the index structure in text files.

We begin with building the path-based mapping together with the dewey encoding for the given XML data. The XML data can be downloaded at:

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/BIOMD0000000009.xml>

The following paper describes how to use dewey encoding (a detailed example is available at the end of this section shown in Figure 1):

<https://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/StoringAndQueryingOrderedXMLUsingARelationalDatabaseSystem.pdf>

In this exercise, we will use the dewey encoding to store the positions of the nodes, (i.e. every node in the XML tree will be assigned a unique dewey_pid). For each unique path of the XML tree, you need to generate three text files: one for the tags (tag file), one for the texts (text file), and one for the attributes and values (attribute file). Tag files consist of the pairs of the tag names together with their dewey_pids with the format as follows.

```
...  
1.1.3.4 tagname1  
1.1.3.5 tagname2  
...
```

Text file has a similar structure, with pairs of text values together with the dewey_pids. Attribute file consists of the triple with the format as follows.

```
...  
1.1.3.4 attributename1 attributevalue1  
1.1.3.5 attributename2 attributevalue2  
...
```

To generate the files, you will use the standard XML event-based parser SAX (read about it, if you are not familiar with it already). Two Java files are prepared for you to be completed. The files can be downloaded at:

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/XMLCounter.java>

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/ReadXMLFile.java>

Your task is to add the content to the functions in *XMLCounter.java*, such as *startElement*, *endElement*. In order to generate the paths and the dewey code, you might need data structures such as Stack and Vector. To complete this task you may use any Java IDE such as Eclipse.

¹ Tetralith: <https://www.nsc.liu.se/systems/tetralith/>

² Apache Hadoop 2.7.0: <https://hadoop.apache.org/docs/r2.7.0/>

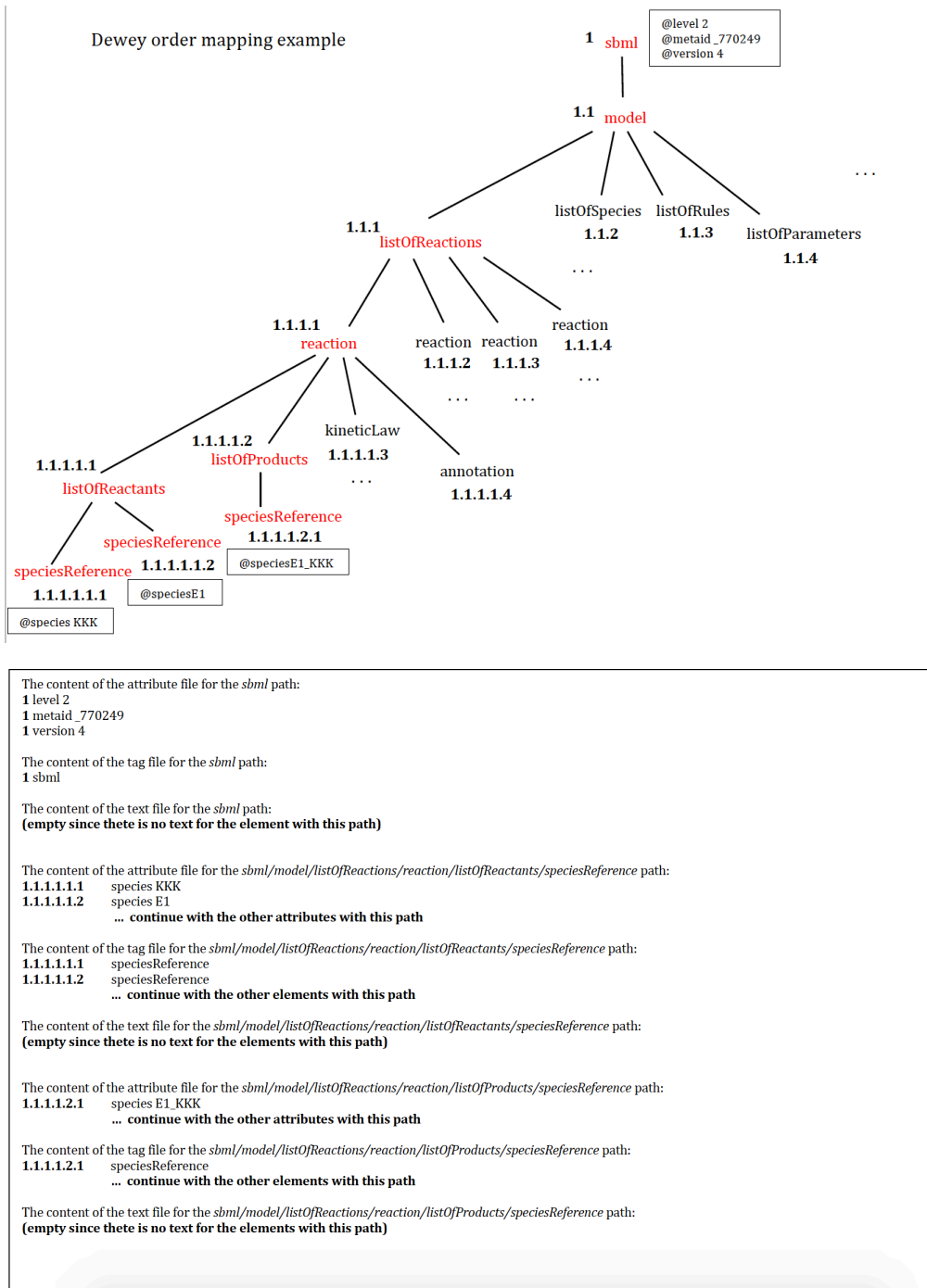


Figure 1: Dewey order mapping Example

1.2 XPath query evaluation

(You can also start to do section 2.1 and 2.2, then get back to this section.)

The purpose of this part is to use the MapReduce programming model to evaluate the following XPath query. Consider the following XPath query over the given XML data:

```
//reaction[listOfProducts/speciesReference[contains(@species, "P_KK")]]/listOfReactants/speciesReference/@species
```

Intuitively, the query asks for the reactant species of the reactions which consist of a product species contain the substring 'P_KK'.

The query can be evaluated by first retrieving the attribute file (file01) with the path:

[//reaction/listOfProducts/speciesReference](#)

and another attribute file (file02) with the path:

[//reaction/listOfReactants/speciesReference](#)

To simplify the task, you do not need to implement the pattern matching steps of the paths. From the example XML data we know there is only one path containing:

[“reaction/listOfProducts/speciesReference”](#) (resp. [“reaction/listOfReactants/speciesReference”](#)).

Thus from all the files you have generated in the last task, there is one attribute file (file01) for the first path and one (file02) for the second path. You need to remember the file names and give them as input parameters for the execution of the program.

Note that the files contain all the products (resp. reactants) from all the reactions. Now we need to conduct a join operation on both data sets by grouping the products and reactants from the same reaction. Therefore, the join attribute is the dewey_pid of the path [//reaction](#). Given a dewey_pid (id) from file01 or file02, we can easily obtain the dewey_pid of [//reaction](#) by removing the last two digits from id. For instance, if the dewey_pid from file01 is [1.1.3.4](#), then the dewey_pid of [//reaction](#) would be [1.1](#).

The join operation is realized by the following two map tasks:

In map task 1 (works on the first part of the query [\[listOfProducts/speciesReference\[contains\(@species,"P_KK"\)\]\]](#)), you need to take a triple from file01, check whether the attribute name is “species” and the attribute value contains the string “P_KK”. If this is true, retrieve the prefix of the current dewey_pid and send it as the key to the reduce task.

In map task 2 (works on the second part [/listOfReactants/speciesReference/@species](#)), what you need to do is similar to Map task 1.

The function in the reduce task is then straightforward.

We prepared the necessary Java files for you to fill up the contents. The files are available at:

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/TextPair.java>

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMain.java>

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMapper1.java>

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinMapper2.java>

<http://www.ida.liu.se/~TDDD43/themes/themeNOSQLlabs/code/JoinReducer.java>

You will need all the files to complete the task. From the above files, *TextPair.java* and *JoinMain.java* are complete and you should not change them. Your task is to complete the other three files.

There are two Mappers and each one of them sends the intermediate <key, value> pairs to the reduce task, where the pairs with the same key are sent to the same reduce task. At the reducer side, there should be mechanism to identify for a received data pair, which Mapper does it come from. The solution is to tag the pairs in the Mapper side with some distinct number.

This kind of join operation is known as Reduce-side Join.

2. Running Java Hadoop Program on Tetralith

2.1 Working on Tetralith

The Tetralith server is available at `tetralith.nsc.liu.se` (log in using your NSC accounts).

To log in Tetralith, please use the following command in the terminal from a computer in the lab rooms:

- `ssh -X username@tetralith.nsc.liu.se` where username is your NSC username (not the LiU one), `-X` indicates forwarding function of `ssh` which is used for running graphics applications remotely. When you logout the server, if the logout is hung, please use `ctrl-c` to terminate the connection.
- You can use Emacs for coding remotely by running `emacs &` in the terminal after you connect to tetralith or program locally, then use `scp` to copy your files to Tetralith.
- `[username@tetralith1 ~]$ emacs &`
- `scp LOCAL_FILE username@tetralith.nsc.liu.se:`

We will use a non-interactive way to run the program on Tetralith. So each time when you want to run your java program, you need to submit your batch job. After that, the job will enter the scheduling queue, where it may have to wait a while until nodes are available to tun the job. We use `sbatch` command to submit the batch job and `squeue` command to monitor your submitted job. You may also use `scancel` command to cancel a job.

- `[username@tetralith1 ~]$ sbatch run.q`
- `[username@tetralith1 ~]$ squeue -u username`
- `[username@tetralith1 ~]$ scancel JOBID`

The Tetralith uses `Slurm` for scheduling. Once you submit a job, the job will be assigned an ID. After the job is finished, you will see a `slurm-ID.out` file returned, which includes the output information of the job script.

We provide a script named `compile.sh` to compile your java code and a script named `run.q` to run the program. Please read the scripts carefully before you change it. For compiling your java code, you don't have to submit the job for scheduling. You can simply run the script as follows.

- `[username@tetralith1 ~]$./compile.sh`

Notice: NSC reserves 6 compute nodes for the two lab sessions in TDDD43 which means other jobs on Tetralith will not use these nodes during our lab sessions. To use the reservations, you can use following commands.

- `[username@tetralith1 ~]$ listreservations`
`Reservations available to user:username / project(s):snic2019-7-77`
`tddd43-2019-12-03 from 2019-12-03T10:15:00 to 2019-12-03T13:00:00 (project:snic2019-7-77)`
`tddd43-2019-12-10 from 2019-12-10T10:15:00 to 2019-12-10T13:00:00 (project:snic2019-7-77)`
`devel from NOW to INF (everyone)`
- `[username@tetralith1 ~]$ export SBATCH_RESERVATION=tddd43-2019-12-DD`
- `[username@tetralith1 ~]$ export SBATCH_RESERVATION=devel (Outside of lab time)`

For those of you may be involved in multiple projects on Tetralith, you have to specify which one to use when submitting a job using `sbatch`.

- `[username@tetralith1 ~]$ sbatch -A snic2019-7-77 run.q`

2.2 Running Word Count Example on Hadoop

Step 1: Log in to Tetralith server.

Step 2: There is an example for how to run Word Count program with Hadoop on Tetralith. The path for the example is `/software/sse/manual/spark/examples/hadoop/2_java_wordcount_1.0/`. You can copy this folder to your home folder by using `cp` command.

Step 3: Before you run the program on Hadoop, you need to first compile the code by running the script `compile.sh`. The script is used to Compile `WordCount.java` and create a jar as shown in Figure 3.

Step 4: After the jar file is created, you can use `sbatch` to submit the job. Then you can use `squeue -u USER_NAME` to check your job. Once the job is finished, you can find a returned file named `slurm-ID.out`.

```
x_huali@tetralith1:~/2_java_wordcount_1.0
File Edit View Search Terminal Help
huali50@seriella1:~$ ssh -X x_huali@tetralith.nsc.liu.se
x_huali@tetralith.nsc.liu.se's password:
Last login: Mon Nov 25 09:38:33 2019 from seriella1.ida.liu.se
Welcome to NSC and Tetralith!

**** Project storage directories available to you:
/proj/roarc/users/x_huali
/proj/tddd43/users/x_huali

**** Documentation and getting help:
https://www.nsc.liu.se/support/systems/tetralith-getting-started/
https://www.nsc.liu.se/support

**** Useful commands
To see your active projects and CPU time usage: projinfo
To see available disk storage and usage: snicquota
To see your last jobs: lastjobs
Login to compute node to check running job: jobsh

To tweak job priorities, extend timelimits and reserve nodes: see
https://www.nsc.liu.se/support/batch-jobs/boost-tools/

(Run "nsc-mute-login" to not show this information)

[x_huali@tetralith1 ~]$ listreservations
Reservations available to user:x_huali / project(s):snic2019-7-77,snic2019-3-236
tddd43-2019-12-03 from 2019-12-03T10:15:00 to 2019-12-03T13:00:00 (project:snic2019-7-77)
tddd43-2019-12-10 from 2019-12-10T10:15:00 to 2019-12-10T13:00:00 (project:snic2019-7-77)
devel from NOW to INF (everyone)

Note: set one of the above as default by running:
userreservation RESERVATIONNAME
Or without the userreservation alias:
source /software/tools/bin/userreservation.sh RESERVATIONNAME
[x_huali@tetralith1 ~]$ export SBATCH_RESERVATION=devel
[x_huali@tetralith1 ~]$ ls
Desktop Documents Downloads MapReduceLab Music Pictures Public Templates Videos
[x_huali@tetralith1 ~]$ cp -r /software/sse/manual/spark/examples/hadoop/2_java_wordcount_1.0/ ./
[x_huali@tetralith1 ~]$ ls
2_java_wordcount_1.0 Documents MapReduceLab Pictures Templates
Desktop Downloads Music Public Videos
[x_huali@tetralith1 ~]$ cd 2_java_wordcount_1.0/
[x_huali@tetralith1 2_java_wordcount_1.0]$ ls
WordCount.java compile.sh input run.q
[x_huali@tetralith1 2_java_wordcount_1.0]$ ./compile.sh
Compiling Wordcount Program...
WARNING: log4j.properties is not found. HADOOP_CONF_DIR may be incomplete.
[x_huali@tetralith1 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class WordCount.class compile.sh run.q
WordCount$TokenizerMapper.class WordCount.java input wordcount.jar
[x_huali@tetralith1 2_java_wordcount_1.0]$ sbatch -A snic2019-7-77 run.q
Submitted batch job 5902382
[x_huali@tetralith1 2_java_wordcount_1.0]$ squeue -u x_huali
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
5902382 tetralith run.q x_huali CF 0:02 2 n[797,827]
[x_huali@tetralith1 2_java_wordcount_1.0]$ squeue -u x_huali
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
[x_huali@tetralith1 2_java_wordcount_1.0]$ ls
WordCount$IntSumReducer.class WordCount.class compile.sh run slurm-5902382.out
WordCount$TokenizerMapper.class WordCount.java input run.q wordcount.jar
```

Figure 2: Steps to run word count program

```

x_huali@tetralith1:~/2_java_wordcount_1.0
File Edit View Search Terminal Help
#!/bin/bash

module load spark/.2.4.3-hadoop-2.7-nsc1.lua

export HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar

echo "Compiling Wordcount Program..."
hadoop com.sun.tools.javac.Main WordCount.java
jar cf wordcount.jar WordCount*.class
~
~
"compile.sh" 9L, 234C 1,1 All

```

Figure 3: compile.sh for word count program

```

x_huali@tetralith1:~/2_java_wordcount_1.0
File Edit View Search Terminal Help
#!/bin/bash
#SBATCH --time=10:00
#SBATCH --nodes=2
#SBATCH --exclusive

echo "START AT: $(date)"

module add spark/.2.4.3-hadoop-2.7-nsc1.lua

# Start with clean config
rm -rf run
mkdir run
cd run

# Startup hadoop filesystem and yarn
hadoop_setup

# Run the hadoop example
echo "Prepare output and input directories and files..."
hadoop fs -mkdir -p "hadoop-examples/wordcount" "hadoop-examples/wordcount/input"
hadoop fs -test -d "hadoop-examples/wordcount/output"
if [ "$?" == "0" ]; then
    hadoop fs -rm -r "hadoop-examples/wordcount/output"
fi
hadoop fs -put -f ../input/file* "hadoop-examples/wordcount/input"

echo "Running wordcount program..."
hadoop jar ../wordcount.jar WordCount "hadoop-examples/wordcount/input" "hadoop-examples/wordcount/output"
echo "===== FINAL WORDCOUNT OUTPUT ====="
hadoop fs -cat "hadoop-examples/wordcount/output"/*
echo "===== "

# Shut down yarn and hadoop
hadoop_stop

echo "END AT: $(date)"
~
~
"run.q" 36L, 1041C 1,1 All

```

Figure 4: run.q script for word count program

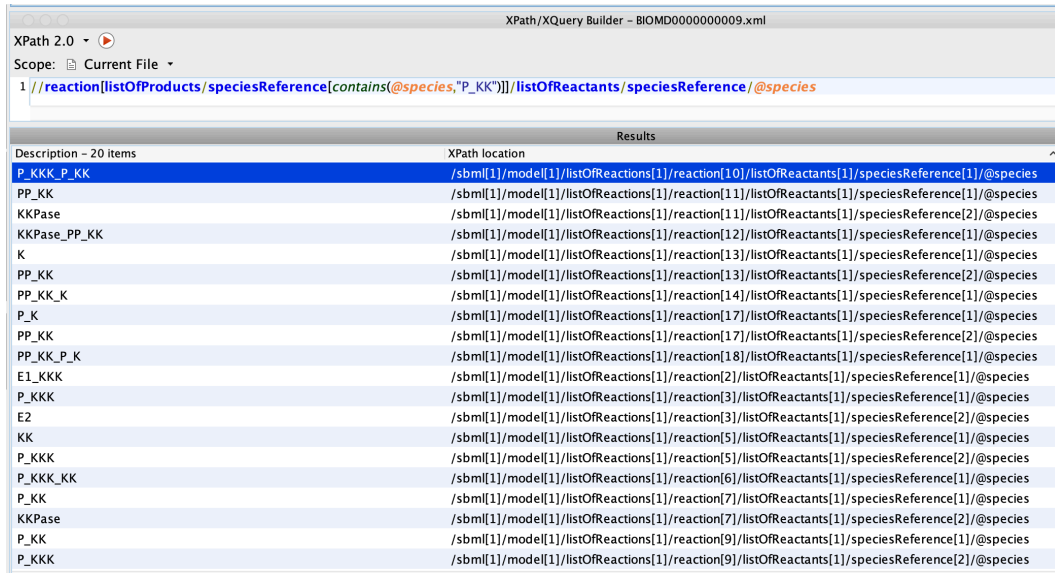
In *run.q* as shown in Figure 4, there are a number of commands that are used to interact with HDFS.

- `hadoop fs -mkdir <FOLDER_NAME>` -make a folder on HDFS
- `hadoop fs -mkdir -p <FOLDER_NAME> <FOLDER_NAME>` -make multiple folders
- `hadoop fs -test -d <FOLDER_NAME>` -if the path is a directory, return 0
- `hadoop fs -rm -r <FOLDER_NAME>` -deletes the directory and any content under it recursively
- `hadoop fs -put <localsrc> ... <dst>` -copy single src, or multiple srcs from local to HDFS
- `hadoop fs -cat <FOLDER_ON_HDFS> [local]` -copy HDFS path to stdout

2.3 Running XPath evaluation program on Hadoop

After you have achieved correct results for section 1.1 and have written correct code for section 1.2, you can prepare to run your program on Hadoop. To compile the Java code and to run the program are similar to step 3 and step 4 in section 2.2, respectively. Indeed, you have to change some details in the two scripts, *compile.sh* and *run.q*.

Figure 5 shows the XPath query result in Oxygen, you are supposed to obtain the same 20 records in your program.



The screenshot shows the Oxygen XPath 2.0 interface. The title bar reads "XPath/XQuery Builder - BIOMD000000009.xml". The "Scope" is set to "Current File". The XPath query is: `1 //reaction[listOfProducts/speciesReference[contains(@species,"P_KK")]]/listOfReactants/speciesReference/@species`. The results table has two columns: "Description - 20 Items" and "XPath location".

Description - 20 Items	XPath location
P_KKK_P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[10]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[11]/listOfReactants[1]/speciesReference[1]/@species
KKPase	/sbml[1]/model[1]/listOfReactions[1]/reaction[11]/listOfReactants[1]/speciesReference[2]/@species
KKPase_PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[12]/listOfReactants[1]/speciesReference[1]/@species
K	/sbml[1]/model[1]/listOfReactions[1]/reaction[13]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[13]/listOfReactants[1]/speciesReference[2]/@species
PP_KK_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[14]/listOfReactants[1]/speciesReference[1]/@species
P_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[17]/listOfReactants[1]/speciesReference[1]/@species
PP_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[17]/listOfReactants[1]/speciesReference[2]/@species
PP_KK_P_K	/sbml[1]/model[1]/listOfReactions[1]/reaction[18]/listOfReactants[1]/speciesReference[1]/@species
E1_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[2]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[3]/listOfReactants[1]/speciesReference[1]/@species
E2	/sbml[1]/model[1]/listOfReactions[1]/reaction[3]/listOfReactants[1]/speciesReference[2]/@species
KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[5]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[5]/listOfReactants[1]/speciesReference[2]/@species
P_KKK_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[6]/listOfReactants[1]/speciesReference[1]/@species
P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[7]/listOfReactants[1]/speciesReference[1]/@species
KKPase	/sbml[1]/model[1]/listOfReactions[1]/reaction[7]/listOfReactants[1]/speciesReference[2]/@species
P_KK	/sbml[1]/model[1]/listOfReactions[1]/reaction[9]/listOfReactants[1]/speciesReference[1]/@species
P_KKK	/sbml[1]/model[1]/listOfReactions[1]/reaction[9]/listOfReactants[1]/speciesReference[2]/@species

Figure 5: XPath query result in Oxygen

Hand in:

To demonstrate the lab, please e-mail a lab report containing all code you have written for both section 1.1 and section 1.2, and also include a file to show the result of your program.