Assignment 3 - Analysis of Dependency Extraction Methods

Team C-Lion

Electrical Engineering and Computer Science, York University

EECS 4314 - Advanced Software Engineering

Dr. Zhen Ming Jiang

December 6st, 2021

Group Member:

Yonis Abokar, Qasim Ahmed, Chandler Cabrera, Taswar Karim, Amir-Hossein Khademi, Chirag Sardana, Sarwat Shaheen

Table Of contents

1.0 - Abstract

This report analyzes and compares different methods of dependency extraction. The first method, Understand, is a tool used in a previous assignment. The Include method was a script created by the group to extract dependencies based on the *include* directive at the top of each C file and also dependencies existing using function calls. SrcML is a tool found online purpose-built for dependency extraction. Two types of analysis were performed: quantitative analysis compares the sets of results of each extraction method with one another, while the qualitative analysis compares the usefulness and accuracy of each method. Finally, the strengths and weaknesses of each extraction method are detailed.

2.0 - Overview of Extraction Techniques

2.1 - Understand

Understand is an Integrated Development Environment (IDE) created by Scientific Toolworks. It's purpose is to allow the user to perform a Static Program Analysis on software and supports the analysis of many programming languages. It's main functionality is to find the dependencies between each file of the project by analyzing their relations, whether that is through inheritance, header files, imports, or some other method (this may depend on the language being analyzed). The results of this analysis may be inspected using an array of tools provided by *Understand*. Its main analysis tool is a dependency diagram, but for the purposes of this assignment, its ability to export the results to a Comma Separated File (.csv) was used for further analysis.

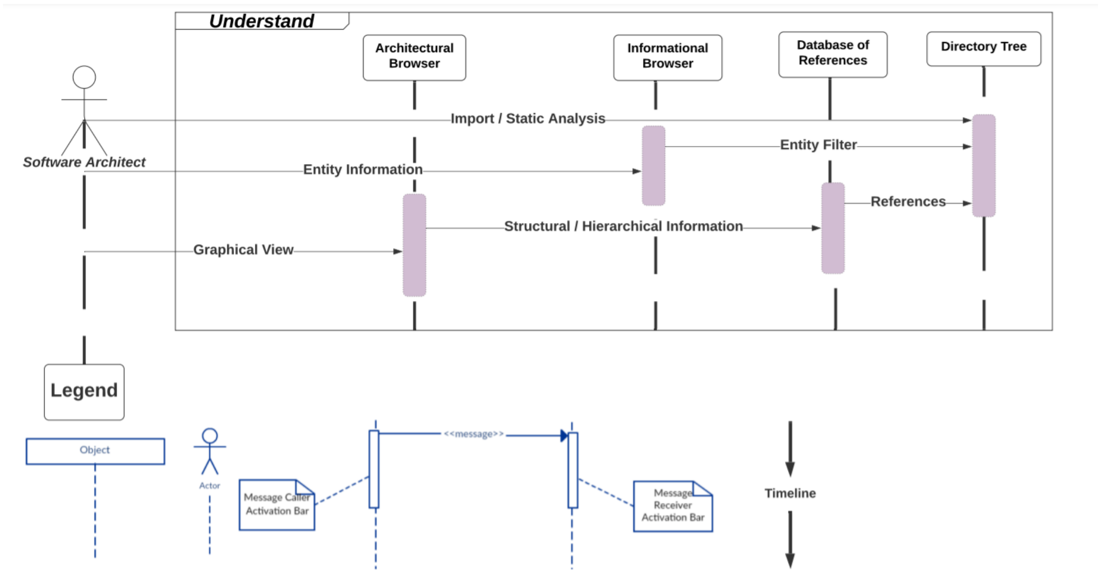


Figure 1: Sequence Diagram for the Understand Dependency Extraction Tool

2.2 - Extract Include Directives using Python

Python scripts were created to statically extract all .c to .h dependencies using the include directive. First the script uses the os library which allows you to get the paths of the folders.Using the os library, the paths under the source code folder for Postgres can be gathered.We start searching for directories and once we locate a folder, we use the os.listdir() method to locate the files under the folder . All the include statements are extracted using the #include statements which have .h in the line . Once all the .h files that are used as dependencies are extracted using the os.path() method then their paths are placed in a list . Secondly we look for all the .c and .cpp files and their paths are placed using the os.path() library in a list. We have to then look inside the .c and .cpp files and create from and to dependencies using the # include statements for .h files . We use the with open(path) method for this purpose and append the file to add all the from and to dependencies in a .txt file . We then use pandas to extract the dependencies and then use the pd.read_csv() method to place the dependencies in a .csv file.

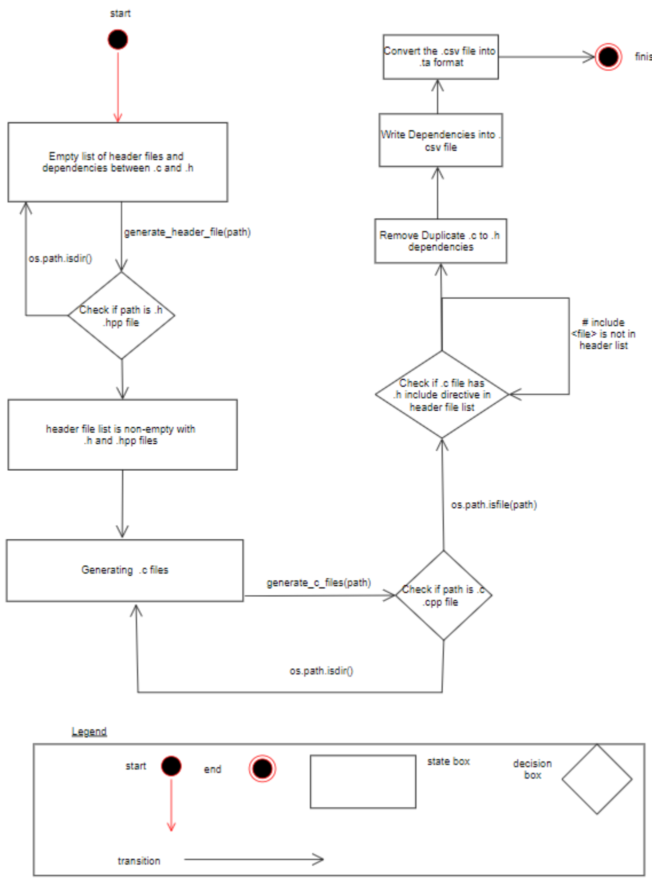

start
Convert the .csv file into .ta format
finish
Empty list of header files and dependencies between .c and .h
Write Dependencies into . csv file
Remove Duplicate .c to .h dependencies
generate_header_file(path)
os.path.isdir()
Check if path is .h .hpp file
# include <file> is not in header list
Check if .c file has .h include directive in header file list
header file list is non-empty with .h and .hpp files
os.path.isfile(path)
Generating .c files
generate_c_files(path)
Check if path is .c .cpp file
os.path.isdir()
Legend
start
end
state box
decision box
transition

Figure 2: The include dependencies extracted_dependencies_srcML.xml

2.3 - srcML

srcML is a free software application that transforms C, C#,C++, Java source code into an XML format representation. The source code can be converted to an XML file which includes different tags hence this provides a new way for the analysis of source code. This application is a command line application that can take directories/files as input, the whole structure is recursively parsed, XML file name can be specified and once the command is executed the XML file gets generated in the desired location where command was generated

The way we generated the XML file is that we went to the folder where the src folder exists and we executed the below command. This command creates a file called src.xml. This is illustrated using the following use case diagram.

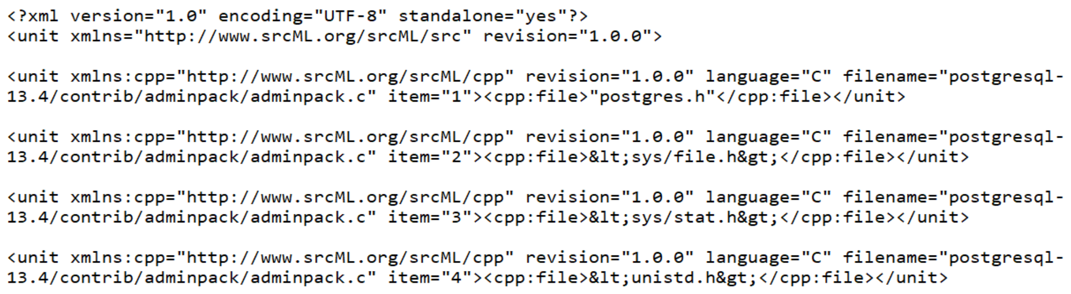
srcml --verbose postgresql-13.4 -o dependencies_srcML.xml

Once the XML was created we used the XML query language to select particular nodes from the XML file(Xpath Query Process in use case diagram) . We used the following command. The // selects everything after the slash in the entire document.

srcml --xpath "//cpp:include//cpp:file" dependencies_srcML.xml

> extracted_dependencies_srcML.xml

Below shows what the extracted_dependencies_srcML.xml looks like.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<unit xmlns="http://www.srcML.org/srcML/src" revision="1.0.0">

<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="1.0.0" language="C" filename="postgresql-
13.4/contrib/adminpack/adminpack.c" item="1"><cpp:file>"postgres.h"</cpp:file></unit>

<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="1.0.0" language="C" filename="postgresql-
13.4/contrib/adminpack/adminpack.c" item="2"><cpp:file>&lt;sys/file.h&gt;</cpp:file></unit>

<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="1.0.0" language="C" filename="postgresql-
13.4/contrib/adminpack/adminpack.c" item="3"><cpp:file>&lt;sys/stat.h&gt;</cpp:file></unit>

<unit xmlns:cpp="http://www.srcML.org/srcML/cpp" revision="1.0.0" language="C" filename="postgresql-
13.4/contrib/adminpack/adminpack.c" item="4"><cpp:file>&lt;unistd.h&gt;</cpp:file></unit>
```

Figure 3: The include dependencies extracted_dependencies_srcML.xml

We also used the two commands below:

srcml --xpath "//src:function/src:name" dependencies_srcML.xml > out1.xml

srcml --xpath "//src:expr/src:name" dependencies_srcML.xml > out2.xml

The first command is picking all the function definitions and in which file they are defined. The second command is looking at which functions are called and in which file the functions are called. We wrote a python script to check into both the out1 and out2.xml. The logic is that the file where the function is called is dependent on the file where the function is defined. Hence we are forming dependencies of files in this way. We just combined the output of extracted_dependencies_srcML.xml and our script to generate the overall raw.ta for srcML**.** In the end we just removed the duplicates using an online parsing tool**.**

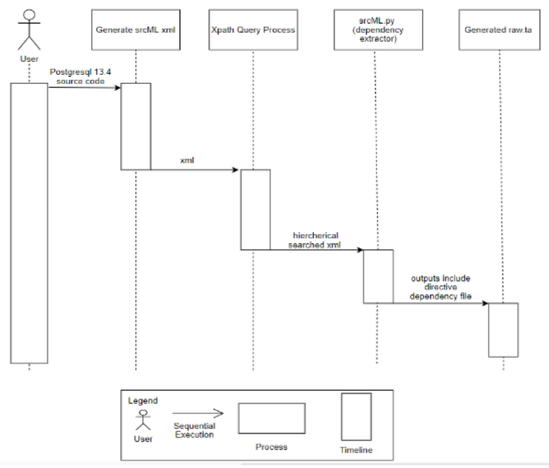
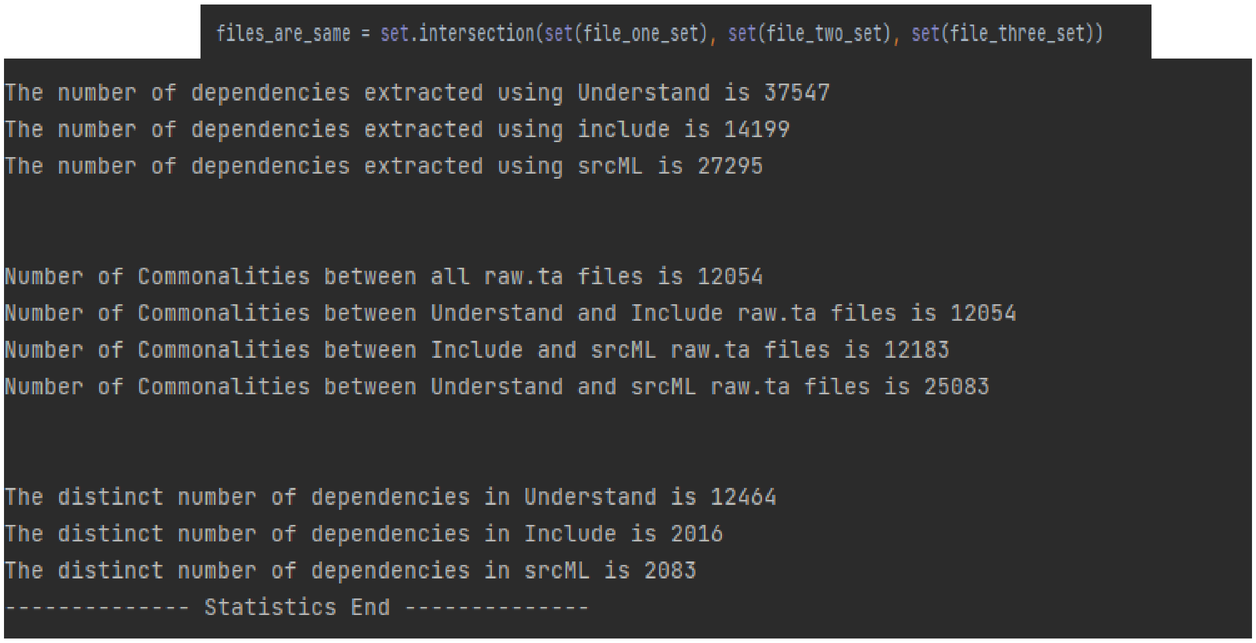

User
Generate srcML xml
Xpath Query Process
srcML.py (dependency extractor)
Generated raw.ta
Postgresql 13.4 source code
xml
hiercherical searched xml
outputs include directive dependency file
Legend
User
Sequential Execution
Process
Timeline

Figure 4: Sequence Diagram for srcML

3.0- Comparison Metrics Process

To compare the dependencies in the raw.ta files we wrote a python program 'Statistics.py' that reads all the raw.ta files into a list data structure and then converts the stored file dependencies into a set where set operations such as set difference can be performed to get the commonalities and differences between all dependency techniques. Ultimately, the script outputs the details that are needed to do analysis by outputting all the comparison numbers between each technique and outputs the the file dependency commonalities and differences in separate raw.ta files for further analysis to understand as to the reason each technique has file dependencies that is not present in the other techniques.



```
files_are_same = set.intersection(set(file_one_set), set(file_two_set), set(file_three_set))
```

```
The number of dependencies extracted using Understand is 37547
The number of dependencies extracted using include is 14199
The number of dependencies extracted using srcML is 27295


Number of Commonalities between all raw.ta files is 12054
Number of Commonalities between Understand and Include raw.ta files is 12054
Number of Commonalities between Include and srcML raw.ta files is 12183
Number of Commonalities between Understand and srcML raw.ta files is 25083


The distinct number of dependencies in Understand is 12464
The distinct number of dependencies in Include is 2016
The distinct number of dependencies in srcML is 2083
-------------- Statistics End --------------
```

Figure 5: The Output of Statistics.py script

4.0 - Comparison of Extraction Techniques

4.1 - Quantitative Analysis

Below are the results of statistics.py shown visually with a brief description. Ultimately there is a large difference in the number of dependencies & common dependencies we found based on our statistics.py, This leads to the fact that there are a lot of discrepancies between the techniques we explained in the earlier part of the report.The results and analysis will be talked about more in depth in the qualitative part of the report along with precision and recall.

4.1.1 Understand Vs. Include

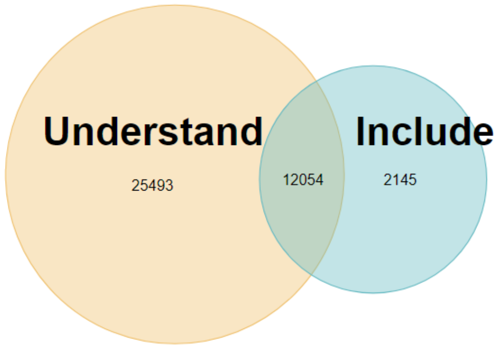


Figure 6: Comparison of the number of dependencies found by Understand vs Include

As it can be seen from figure one, the number of common dependencies found was 12054, 25493 of those are distinct to Understand and 2145 are distinct to include. This means that out of the total number of dependencies found, the script could not locate all of them. This tells us that the Include script is missing something. One reason for this is the fact that the understand tool was not able to read some of the files due to naming conventions.

4.1.2 Include Vs. srcML

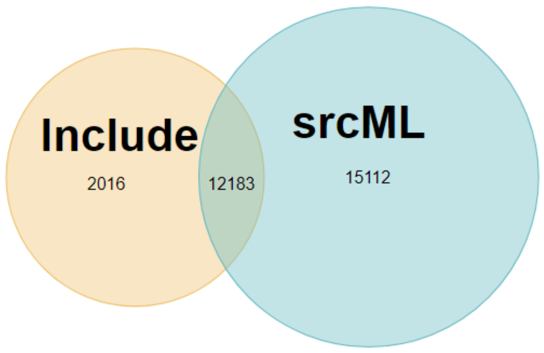


Figure 7: Comparison of the number of dependencies found by Include vs srcML

The total number of dependencies found by SrcML and Include can be seen in the figure 5 with 15112 exclusive to SrcML,12183 common and 2016 exclusive to Include.

4.1.3 Understand Vs. srcML

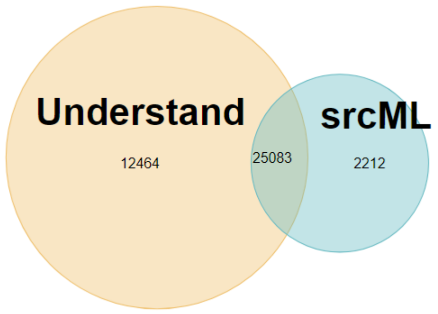
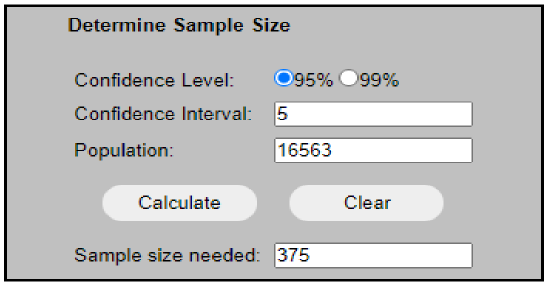


Figure 8: Comparison of the number of dependencies found by Understand vs srcML

Figure 6 illustrates the differences between Understand and srcML. The total number of dependencies found in common was 25083, 12464 was exclusive to Understand and 2212 exclusive to srcML. A majority of the discrepancies between them are likely due to the fact that understand looks at more than just the include headers.

4.2 - Qualitative Analysis

In order to do the qualitative analysis on the comparison we leveraged the sample size calculator [1] used in statistical inference techniques. We used a sample size as the dependencies we have are huge and it is not feasible to manually compare them one by one. We utilized the sample size calculator to calculate the sample size. We used a confidence level of 95%, a confidence interval of 5% with a population of 16563(Total dependencies) giving us a sample size of 375 dependencies which can be seen in the figure below:


Determine Sample Size

Confidence Level: [✓ 95%] 99%

Confidence Interval: 5

Population: 16563

Sample size needed: 375

Figure 9: Calculation for the sample size

Since we have a sample size of 375, from the Statistics.py we have 16563 distinct dependencies. Thus the percentage of Understand dependencies in sample is
(12464/16563) * 375 => 282 => 282 dependencies for Understand

Similarly we have 2016 distinct dependencies for Include. Thus the percentage of Include dependencies in sample is (2016/16563) * 375 => 45 dependencies for Include

Lastly we have 2083 distinct dependencies for srcML. Hence the percentage of srcML dependencies in sample is (2083/16563) * 375 = 47 => 47 dependencies in srcML

4.2.1 Dependencies found by Understand only

Most dependencies that exist, or were referenced to by other cFiles, within postgresql-13.4/contrib/ were correctly picked up by Understand, whereas the Include/srcML techniques failed to pick up such dependencies. Some of the examples of these dependencies which were not picked up by Include/srcML are provided below:

- postgresql-13.4/contrib/pgcrypto/crypt-des.c
- postgresql-13.4/contrib/postgres_fdw/connection.c
- postgresql-13.4/contrib/fuzzystrmatch/dmetaphone.c
- postgresql-13.4/contrib/pgcrypto/pgp-decrypt.c
- postgresql-13.4/contrib/ltree/ltree.h

Similar examples of the dependencies within postgresql-13.4/contrib/ were possibly not picked by Include/srcML because the contrib archive area contains supplemental packages intended to work with the Postgre-SQL distribution, but which require software outside of the database distribution to either build or function. Most dependencies within contrib contain references to wrapper packages or other sorts of accessories required by non-free programs. Since Understand is able to create reference links between entities such as variables, methods, or even classes and structs, it is better at picking up such dependencies, whereas tools like Include/srcML are not.

- Macro Use

postgresql-13.4/src/backend/commands/tablecmds.c→postgresql-13.4/src/include/pg_config_manual.h

```
Relation    pkrel;
int16       pkattnum[INDEX_MAX_KEYS];
int16       fkattnum[INDEX_MAX_KEYS];
Oid             pktypoid[INDEX_MAX_KEYS];
Oid             fktypoid[INDEX_MAX_KEYS];
```

This dependency is only recognized by Understand. Understand recognizes this based on a macro that is defined in pg_config_manual.h and used in tablecmds.c. Since Understand is the only approach that recognizes macro uses, this dependency is distinct to Understand only.

- Type referencing

postgresql-13.4/contrib/adminpack/adminpack.c→postgresql-13.4/src/backend/access/common/tupdesc.c

```
static Datum
pg_logdir_ls_internal(FunctionCallInfo fcinfo)
{
	ReturnSetInfo *rsinfo = (ReturnSetInfo *) fcinfo->resultinfo;
	bool		randomAccess;
	TupleDesc	tupdesc;
	Tuplestorestate *tupstore;
    ...
}
```

The above dependency is only found using Understand. Instead of using #include, this dependency was discovered by Understand due to an object declaration of type *TupleDesc(shown in red above)* in the source code. In the "adminpack.c" file. Due to the lack of #include in this dependency, this dependency could not have been discovered by the Include approach.

- Type casting

postgresql-13.4/contrib/pg_freespacemap/pg_freespacemap.c→postgresql-13.4/src/backend/storage/freespace/freespace.c

```
freespace = GetRecordedFreeSpace(rel, blkno);
relation_close(rel, AccessShareLock);

PG_RETURN_INT16(freespace);
```

In this dependency, the client casted freespace into a variable type *PG_RETURN_INT16*, which is found in "freespace.c"

- Function call

postgresql-13.4/src/backend/main/main.c →postgresql-13.4/src/backend/bootstrap/bootstrap.c

```
if (argc > 1 && strcmp(argv[1], "--boot") == 0)
	AuxiliaryProcessMain(argc, argv);	/* does not return */
```

In this dependency, the client code did not have "#include bootstrap.h" in its code. Instead, it made a direct function call to a function *AuxiliaryProcessMain()* that is part of "bootstrap.c". Since this is allowed in C++, it created a dependency that did not require the use of #include. Understand was able to pick up on this dependency, but the other two methods did not.

- Variable referencing

postgresql-13.4/contrib/pg_buffercache/pg_buffercache_pages.c→postgresql-13.4/src/include/utils/elog.h

```
elog(ERROR, "return type must be a row type");
```

This dependency was established by "pg_buffercache_pages.c" making a function call and passing a *ERROR* parameter that belongs to "elog.h". Again, no #include was used in this context. Note that the the function *elog()* is also a function defined in "elog.h". Calling this function also establishes a dependency between the two files.

4.3.2 Dependencies found by Include only

(*) → postgresql-13.4/src/include/port/win32_port.h

- *(*) represents any file that depends on win32_port.h. These are:* postgresql-13.4/contrib/auth_delay/auth_delay.c, postgresql-13.4/src/backend/replication/basebackup.c, postgresql-13.4/src/backend/utils/misc/pg_config.c, and postgresql-13.4/src/bin/pg_config/pg_config.c
- Any of the dependencies with this form that were uniquely discovered by the *Include* method are transitive dependencies (A → B and B → C, then A → C) that occur if on a Windows system

postgresql-13.4/contrib/hstore_plpython/hstore_plpython.c → postgresql-13.4/src/include/storage/bufmgr.h

- This is an example of dependency found by Include
- After carefully checking the file hstore_plpython.c it was discovered that Include picked this bufmgr.h header file by mistake and that hstore_plpython.c is not dependent on the bufmgr.h header file.
- It has been seen in the Include method that generally the c files depending on the file bufmgr.h are wrong dependencies that Include method picked. In other words most of these c files do not depend on bufmgr.h.

postgresql-13.4/contrib/auth_delay/auth_delay.c → postgresql-13.4/src/include/port/win32_port.h

- This is an example of a transitive dependency only found by Include Method
- Our include script is able to recognize the #include inside conditional directives, this example uses a conditional directive as shown below.
```
#if defined(WIN32) && !defined(CYGWIN)
    #include "port/win32_port.h"
#endif
```
- Both of the other techniques were not able to get this dependency. The reason why Understand was not able to grab this dependency is because Understand doesn't look at #include in conditional directives. The srcML is not able to find this dependency because srcML only finds a few transitive dependencies and doesn't find all the transitive dependencies. srcML however does look at #include in conditional directives.

Something interesting that has been noted for Include and both the other techniques is that if an #include header file is not defined in postgresql-13.4 paths and is an external C library then none of the techniques are able to grab the dependencies that rely on these external C libraries. This accounts for a small number of cases.

4.2.3 Dependencies found by srcML only

As mentioned above the distinct dependencies found by srcML is 46. Based on the calculation above to be on the safe side we will look at more than 1 dependency in srcML.The following are just a few examples of some dependencies in srcML only.

postgresql-13.4/src/backend/optimizer/util/tlist.c postgresql-13.4/src/common/fe_memutils.c
postgresql-13.4/src/backend/utils/mmgr/aset.c postgresql-

13.4/src/backend/utils/mmgr/memdebug.c

For the dependency, **postgresql-13.4/src/backend/optimizer/util/tlist.c postgresql-13.4/src/common/fe_memutils.c**, the parent of an entity is determined during analysis by the parser and is language specific. There may be many unneeded dependency references from **/src/backend/optimizer/util** to CFiles and header files commonly present in other directories, such as **common,** within PostgreSQL source code. Some of the feature calls made from the **util** folder are also picked up as dependency references, even though most of these feature calls are made use of during runtime, as commonly referred to as conditional dependencies. More sophisticated tools, such as Understand, are able to exclude these conditional feature call references from its generated dependencies.

When talking about the dependency **postgresql-13.4/src/backend/utils/mmgr/aset.c postgresql-13.4/src/backend/utils/mmgr/memdebug.c** The function void
randomize_mem(char *ptr, size_t size) is defined in memdebug.c and aset.c file is calling the function randomize_mem. This was picked up by srcML as srcML is able to pick dependencies at function level and where they are defined. This is not picked up by Understand in this case as Understand is not looking at dependencies based on declared functions/function calls. It looked at dependencies in this particular case based on macros and #include directives. Include was able to pick up the dependency postgresql-13.4/src/backend/utils/mmgr/aset.c postgresql-13.4/src/include/utils/memdebug.h but it was not smart enough to recognize that memdebug.c depends on memdebug.h and hence aset.c depends on memdebug.c

4.2.2 Comparison of Dependencies srcML vs Understand

The following shows a few examples of dependencies found in srcML that are not found in Understand and their reasons

postgresql-13.4/contrib/fuzzystrmatch/dmetaphone.c⟶ postgresql-13.4/src/include/common/string.h

- This is an example of a dependency that is picked up by srcML.
- The c file dmetaphone.c is picked up by Understand but Understand didn’t catch the header file postgresql-13.4/src/include/common/string.h.
- The reason why Understand was not able to pick up this string.h header file is because the file dmetaphone.c has #include in the start of the file and then there is a define

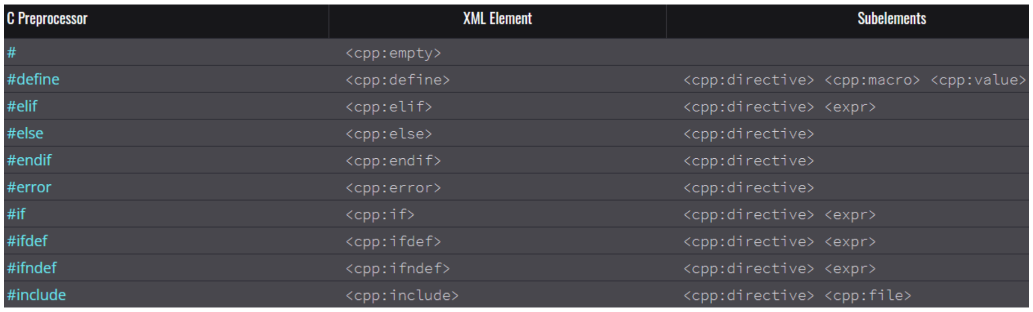
  directive used followed by #else directive. All the .h files in the #else directive have been ignored by Understand. Below is part of the dmetaphone.c file.

  ```
  #include "postgres.h"
  #include "utils/builtins.h"
  #define NDEBUG
  #else
  #include <stdio.h>
  #include <stdlib.h>
  #include <string.h>
  #include <stdarg.h>
  #endif
  ```

postgresql-13.4/src/backend/libpq/be-secure-openssl.c→ postgresql-13.4/src/include/port/win32/arpa/inet.h

- This is again an example of dependency that is picked up by srcML.
- The c file be-secure-openssl.c is again picked by Understand but Understand failed to pick the header file inet.h
- When we examined the be-secure-openssl.c, the reason why Understand was not able to pick up this inet.h header file is because Understand doesn't recognize the #ifdef directive and hence it ignores the .h files inside the if directive. Below are the directives of the be-secure-openssl.c file.

```
#include "postgres.h"
#ifdef HAVE_NETINET_TCP_H
#include <netinet/tcp.h>
#include <arpa/inet.h>
#endif
```



C Preprocessor	XML Element	Subelements
#	<cpp:empty>	
#define	<cpp:define>	<cpp:directive> <cpp:macro> <cpp:value>
#elif	<cpp:elif>	<cpp:directive> <expr>
#else	<cpp:else>	<cpp:directive>
#endif	<cpp:endif>	<cpp:directive>
#error	<cpp:error>	<cpp:directive>
#if	<cpp:if>	<cpp:directive> <expr>
#ifdef	<cpp:ifdef>	<cpp:directive> <expr>
#ifndef	<cpp:ifndef>	<cpp:directive> <expr>
#include	<cpp:include>	<cpp:directive> <cpp:file>

Fig 10: directives explained in srcML

4.2.5 Comparison of Dependencies srcML vs Include

The following shows a few examples of dependencies found in srcML that are not found in Include and their reasons

postgresql-13.4/src/backend/libpq/be-secure.c→ postgresql-13.4/src/include/port/win32/arpa/inet.h

- The above dependency is present in srcML
- Include script was able to pick up the be-secure.c file but it wasn't able to pick up the inet.h header file.
- The reason why Include script was not able to grab the inet.h header file is because it tries to find the header files from the start of file and once it sees anything other than #include it stops and hence only processes the #include lines and not other directives..
- In this particular case again the Include script doesn't know how to read inside #ifdef directive and skips the header files in the #ifdef directive. Below shows most of the directives for the be-secure.c file.

```
#include "postgres.h"
#ifdef HAVE_NETINET_TCP_H
#include <netinet/tcp.h>
#include <arpa/inet.h>
#endif
```

postgresql-13.4/src/include/c.h → postgresql-13.4/src/interfaces/ecpg/test/preproc/strings.h

- The above dependency is again present in srcML as srcML was able to grab transitive dependencies.

- The Include script was able to pick the c.h header file. Include script was not able to grab this specific dependency
- Include script was not able to pick this specific dependency as include script is not grabbing the transitive dependencies. In other words, the Include script is not grabbing the .h → .h dependencies.

4.3 - Precision and Recall

Precision and Recall is a statistical performance measurement which essentially estimates the usefulness and completeness of data. Precision is defined as the proportion of selected items that are relevant whereas, Recall is defined as the proportion of relevant items that are selected.

The chart below examines the quality of the Include and srcML dependency extraction methods by using Precision and Recall, and if we assume that Understand method acts as an oracle which essentially means all the dependencies extracted by understand are all correct and present.

Confidence Level: 95%; Confidence Interval: 5; Total size: 16563; **Sample size = 375**

Method	**Understand**	**srcML**	**Include**
Dependencies in Sample	282	47	45
Percentage in sample	75%	13%	12%
Correct sample dependencies	216	40	13
Ratio of correctness in sample dependencies	77%	86%	29%
Incorrect sample dependencies	66	7	32
Ratio of incorrectness in sample dependencies	23%	14%	71%
Extrapolated correctness(TP)	77% * 12464 = 9597	86% * 2083 = 1791	2016 * 29% = 585
Extrapolate incorrectness(FP)	12464 - 9597 = 2867	2083 - 1791= 292	2016 - 585 = 1431
Precision	TP/(TP+FP) = 9597/12464 = 77%	TP/(TP+FP) = 1604/2083= 86%	TP/(TP+FP) = 585/(585 + 1431) = 29%
Oracle(FN)	$TP_{understand} + TP_{Include} + TP_{srcML} + Commonalities_{understand\&Include} + Commonalities_{Include\&srcML} + Commonalities_{srcML\&Understand} - 2 * Commonalities_{Understand\&Include\&srcML} = 9597 + 1791 + 585 + 12054 + 12183 + 25083 - 2 * 12054 = 37185$		
Recall	TP / (TP + FN) = 9597 / (9597 + 37185) = 21%	TP/(TP + FN) = 1791/(1791 + 37185) = 4%	TP / (TP + FN) =2%

Analysis of Result:
We take a look at the quality of our extraction method below:

As it can be seen in the table above, srcML has a 86% precision which is slightly higher than Understand 77%. This result finding was expected due to the fact that the method extraction for the Include process would produce some Include statements that were surrounded by strangely placed comments which affected our results. Its worth noting that in general the precision of srcML should be bit higher but our implementation only considers “include” statements, however later on our team learned that srcML is also capable of searching for dependencies at both the method level and function level and this would result in an increase in the precision. When it comes to analyzing the recall for each extraction method of srcML and Include, both have the same recall of 84%. This is to be expected as srcML extracts 16578 dependencies compared to Include which extracts 16241. So srcML extracts marginally more by approximately 337, which would not affect the completeness of the representation of dependencies in the sample size.

Implications and recommendations of comparison metrics

Understand method’s precision and accuracy cannot be improved as that is a software that is being used to extract all the dependencies and it is something that we didn’t write. The include method however can be improved to pick up all the transitive dependencies and all the dependencies that exist inside conditional directives. srcML is by far the most precise method of extracting dependencies. This method can be further improved by looking at all of the transitive dependencies and also looking at all the external C library #include directives.

5.0 Potential Risks and Limitations of each Extraction Approach

A major limitation we became liable to was only considering statements that contained the “INCLUDE” keyword. As we investigated deeper in the source code, we discovered dependencies that existed outside of the *include* directive. Dependencies were found both in function level and method level, as well as in forms of inheritance. Therefore, relying solely on the include directive would result in missed dependencies.
Additionally, due to imperfections in the code, there existed dependencies that were created but were never used. Since dependencies can exist via inheritance, it is not always necessary to include the include directive in the code. Therefore, if an extraction method relies only on counting the include directive, it is possible that certain dependencies have been overlooked.

Another limitation we came across during the investigation is the confidence interval and the impact it has on the accuracy of the analysis. A higher interval reduces the sample size and increases efficiency, but reduces accuracy of the results, while a lower interval improves accuracy but is slower to process. For lack of a scientific approach, we selected a confidence interval of 5%, a reasonable number according to conventions.

Some of the risks associated with the analysis lie with the sample size which has been provided by the Sample Size Calculator. Many of the dependencies chosen for analysis were taken at random. Because of the randomness within the data, certain outlier dependencies were not factored into the analysis. Further data science metrics, outside the above Precision and Recall analysis, may provide us with a detailed picture of the efficacy of each of the extraction methods used.

6.0 - Use Cases

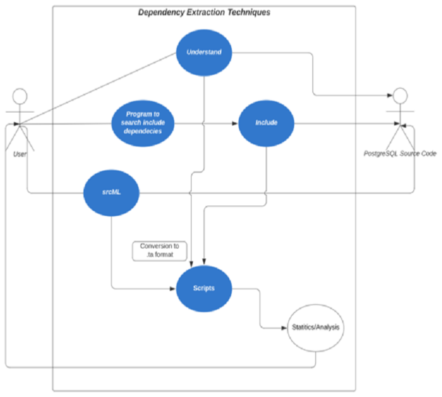


Fig 11: Sequence Diagram for the Understand Dependency Extraction Tool

7.0 Lessons Learned

None of the methods studied so far are perfect. All the methods give slightly different results and have their limitations. We think that there are other techniques that exist that could have been used to do a more thorough analysis for dependency extraction but due to time constraints this report only focuses on 3 techniques. Due to limited time the available people in the group had to pretty much jump and work on all 3 techniques for doing the analysis.

8.0 Conclusion

In conclusion we are able extract the dependencies of PostgreSQL using the three techniques such as srcML, Include and understand and make the qualitative and quantitative to make the appropriate comparison and compare the tools. It was evident that using different dependencies yielded different results and by doing so we increased the accuracy of the results. It's important to note that using different techniques also increased the non valid dependencies which were caused by incorrectly defined dependencies paths and others that could not be found and we had to manually look through to see what was correct and what was not correct.

7.0 - Data Dictionary

FP:	False Positive
FN:	False Negative
TP:	True Positive

9.0 - References

1) https://www.surveysystem.com/sscalc.htm
2) https://documentation.scitools.com/pdf/understand.pdf
3) https://support.scitools.com/support/solutions/articles/70000582792-understanding-understand-dependencies
4) https://doxygen.postgresql.org/

**If images were not clear, please refer to GITHUB for clear images.