Using the CLICS clone detector and GUI

Synopsis

This document will describe how to detect and analyze code clones in a software system using the CLICS clone detector and GUI. This process is broken into 4 steps:

1) Environment setup.
2) Clone detection.
3) Post-process code clones (filter and categorize).
4) Investigate and annotate code clones using user interface.

Note, throughout this document, shell commands will be provided as examples. Each command will begin with `$>' indicating the shell prompt.

This is an initial draft of a help document. It does not yet contain documentation of all of the options for the commands used here.

Step 1: Environment Setup

Part of the clone detection process is the extraction of code regions from within C/C++ source code. These code regions include procedure bodies, macros, and data type definition such as unions and structs. CLICS uses a modified version of ctags to do this and uses the environment variable CTAGS_PATH to find the executable. Before running the clone detection, post-processing, or UI steps be sure to set the path to the modified ctags executable. For example, in bash you would export the variable with the following command (assuming ctags-mod can be found in /usr/local/bin):

```
$> export CTAGS_PATH=/usr/local/bin/ctags-mod
```

Todo: mention lsedit needs to added to your path, mention where to get it.

Step 2: Clone Detection

Clone detection is comprised of three steps: building a list of source files, source code region extraction, and running the clone detector. In the first step, we must construct a file containing a list of source files, one per line. The files should be listed using their full path, not their relative path. This can be conveniently done using find (newer versions of find support more advanced regex operators than the example given here):

```
$> find /home4/cjkapser/Research/cases/pine -name “*.c*” >> pine-files
$> find /home4/cjkapser/Research/cases/pine -name “*.h*” >> pine-files
```

This file will now contain a list of the source files used to build pine. Note in the above example the full path to the source directory is given as the path argument for find.

Next we must extract the regions (procedures, macros, etc.) from the source code. This is done using the command `ExtractRegions.py'. In its typical usage ExtractRegions.py requires two options to be

specified: -o output_file and -f input_filelist. -o specifies the output file that will contain the extracted regions, -f specifies the location of the list of files to extract regions from. Ex:

```
$> ExtractRegions.py -f pine-files -o pine-regions
```

Todo: document region file format.

Once this list of regions is extracted, we can start the clone detection:

```
$> clics_clone_detection -M 2304 -m 60 -o pine-clones -R pine-regions
```

This command takes several options. -M defines the approximate memory to be used to store the suffix tree data structure (the core data structure for locating the code clones). With 4GB total RAM, 3 GB of available RAM on a 32 bit OS, it is generally safe to use 2.25 GB for the suffix tree, leaving ample room for opening files and search for clones. -m specifies the minimum number of lexical tokens that must compose a match for it to be valid. The default is 30 tokens, the above example specifies a minimum of 60 tokens. -o specifies where to place the output of the clone detection process, the above example places the clone output in pico-clones. -R specifies the input file where the extracted regions from the previous step can be found. The complete option summary of clics_clone_detection is:

- -p Parameterize the input (replace identifiers with placeholder and force a semi one-to-one match to filter matches). [conflicts with -e and -r]
- -r Use raw input, do not tokenize or parameterize. [conflicts with -e and -p]
- -e Use tokenized input, do not parameterize. [conflicts with -p and -r]
- -M Amount of memory to use for suffix tree.
- -m Minimum size (in tokens) for an acceptable clone.
- -o Output file [must be specified].
- -R Region file to use to split clones into regions (rather than allow them to cross multiple regions. If not specified, command expects a file containing a list of source files as an argument.

Todo: document clone file contents.

Step 3: Post-processing code clones (filter and categorize)

Once the code clones are detected they need to be filtered, categorized, and loaded into a postgresql database. Ex:

```
$> ConvertToPostgres.py -d pine_clones_60 -f pine-clones
```

This command will create a database `pine_clones_60' and filter it with filtered and classified clones from the set of clones found in the file `pine-clones'. The option -d specifies the database to create, -f specifies the input file. The complete list of options taken by this command are:
[-h host -p port] [-u username] [-P] -d dbname -f filename

- -h hostname of database server
- -p port number the postgresql server is listening on
- -u username to connect to database with

-P ask for a password before connecting
-d database name to create
-f file containing clones to be filtered and loaded into the database.

Todo: document the 6 filters, the categorization, and the database relations.

Summary of Steps 1, 2, and 3

```
$> #Set up environment
$> export CTAGS_PATH=/usr/local/bin/ctags-mod

$> # Make a list of files to detect clones
$> find /home4/cjkapser/Research/cases/pine -name "*.c*" >> pine-files
$> find /home4/cjkapser/Research/cases/pine -name "*.h*" >> pine-files

$> # Extract the region information from the files
$> ExtractRegions.py -f pine-files -o pine-regions

$> # Detect clones using region information
$> clics_clone_detection -M 2304 -m 60 -o pine-clones -R pine-regions

$> # Post process clones and load them into a database
$> ConvertToPostgres.py -d pine_clones_60 -f pine-clones
```

Step 4: Investigate and annotate code clones using user interface

Analyzing clones in software involves inspecting individual clones to manually filter false positives and annotate valid clones. Both of these tasks are highly subjective, and the quality of the process depends very strongly on an understanding of the software being analyzed. It is strongly recommended that design documentation is reviewed and, if necessary, investigate/document the important data structures and sub-sytems before beginning this step. The goal of this software review is to gain an understanding the software system similar to that which you would need to maintain/enhance the software because the decisions you make about code clones should be considered in this context.

Once you have an understanding of the software system, you can open the CLICS user interface with the following command:

```
$> CLICS_GUI.py
```

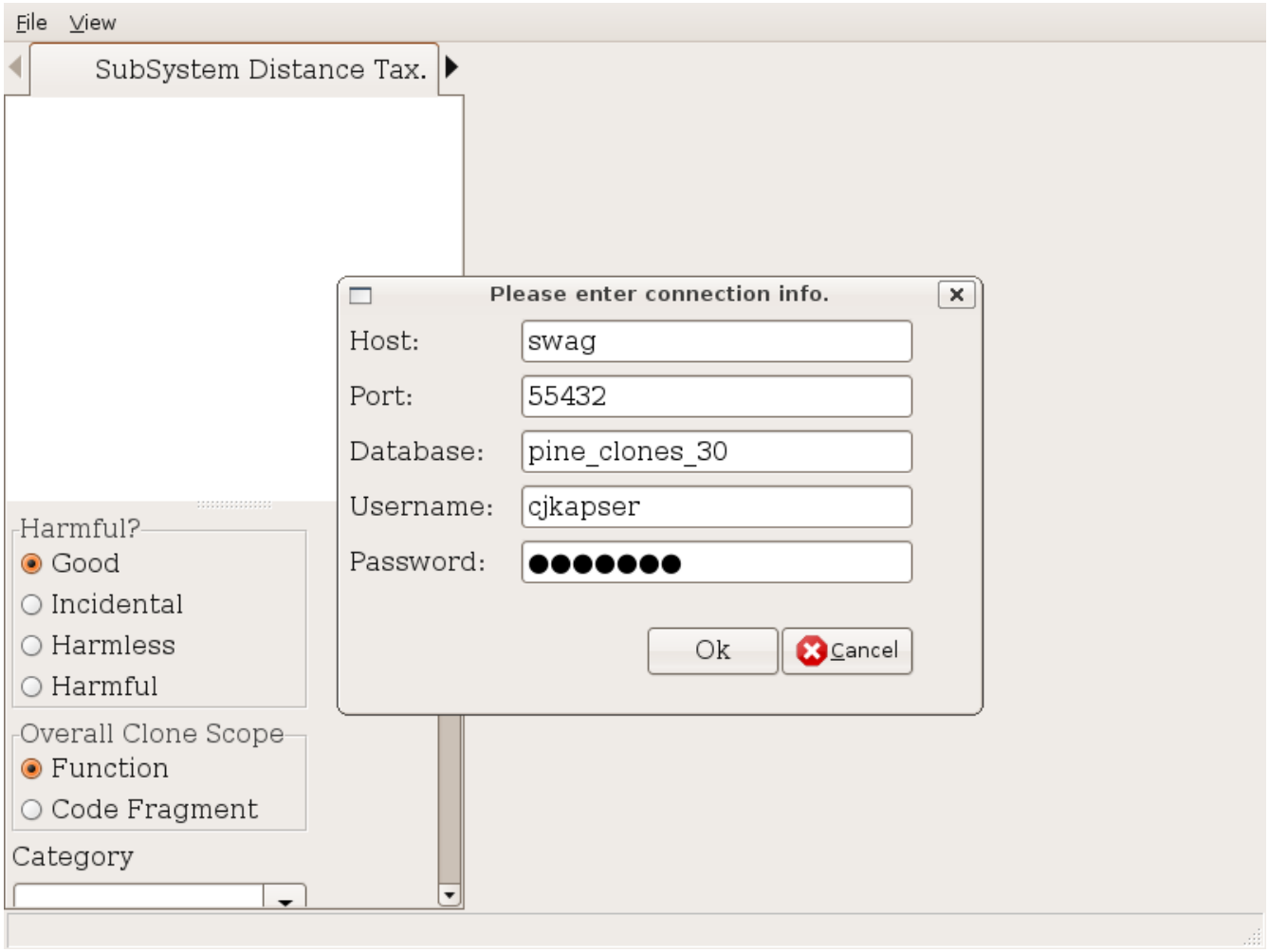

Figure 1: Open Cloning Database

Open the code clone database you created earlier (during post-processing) using the *File → Open DB* menu item. You are presented with a dialog (*Figure 1*). All fields are optional with the exception of *Database*. *Host* is the name of the server hosting the postgresql database, *port* is the port postgresql is listening on, *database* is the name of the database containing the clones, *username* and *password* are the authentication credentials used to connect to the database. *If you are running CLICS on the same machine as the host database, you can often just input the database name.*

The source code used for the analysis may is stored in the database, by default CLICS_GUI will use this source to visualize the clones. The code is stored as a compressed blob to minimize bandwidth usage. If you are connecting to your database over a standard ADSL connection (5Mbps/.6Mbps) viewing the source for the clones should be relatively fast (most of the slowness will be associated with the latency of the database queries themselves).

If you are using a slower network connection you may wish to have the source on your local file-system. This is not what you want in most cases. However, to use use source code located on your local file-system, un-check the *File→Use source in DB* menu item. If you are browsing the code clones on a different machine from the one where the code clone detection was performed, you may need to adjust

the prefix of the file locations. This can be done using the menu item *File → Change Source Dir* (*Figure* 2). The text shown in original path will be replaced by the text in new path. For example, if the path the source files during detection was "/home4/cjkapser/Research/cases/pine/..." and the source location on the machine running CLICS_GUI is "/home/cjkapser/Research/cases/pine" we can replace the beginning of the path with the information shown in Figure 2. This step is required each time you connect to a clone database.

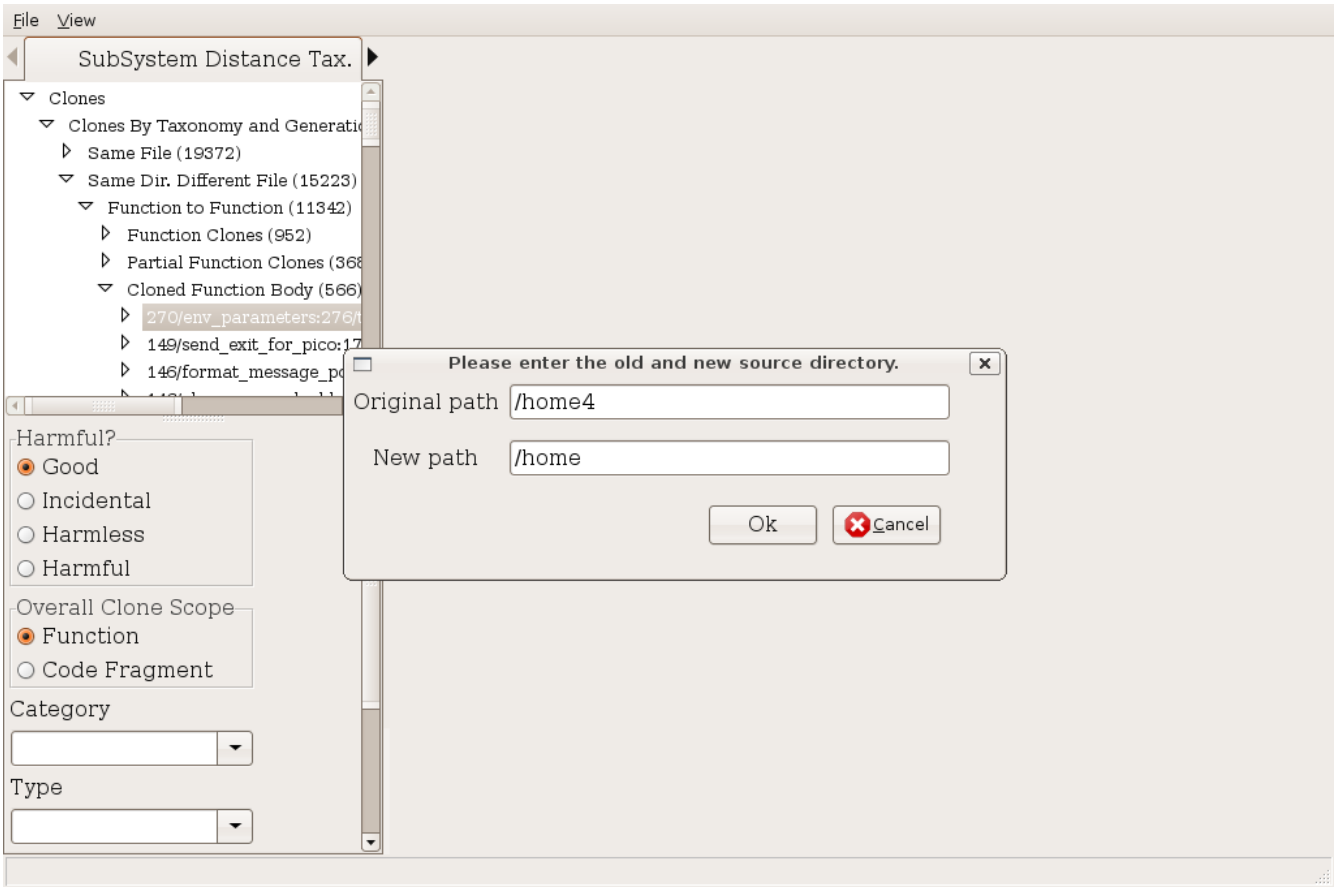

Figure 2: Change Source Directory Prefix

The anatomy of the user interface

The user interface of CLICS provides several views of clones and variety of forms of querying to select or find clones of interest. On the left hand side of the UI you will find tabs allowing you to browse clones by: a simple taxonomy (SubSystem Distance Tax. tab), file-system (Files tab), query results organized by taxonomy (Query tab), a randomly selected clone (Random tab), and by annotated clones (Annotated tab). The rest of this section will discuss each of these tabs and the views of cloning they provide. You will also notice on the bottom left a form that can be used to document selected clones. You will see how to use this in the text that follows. The right hand side of the user interface changes depending on the tab left tab you have currently selected.

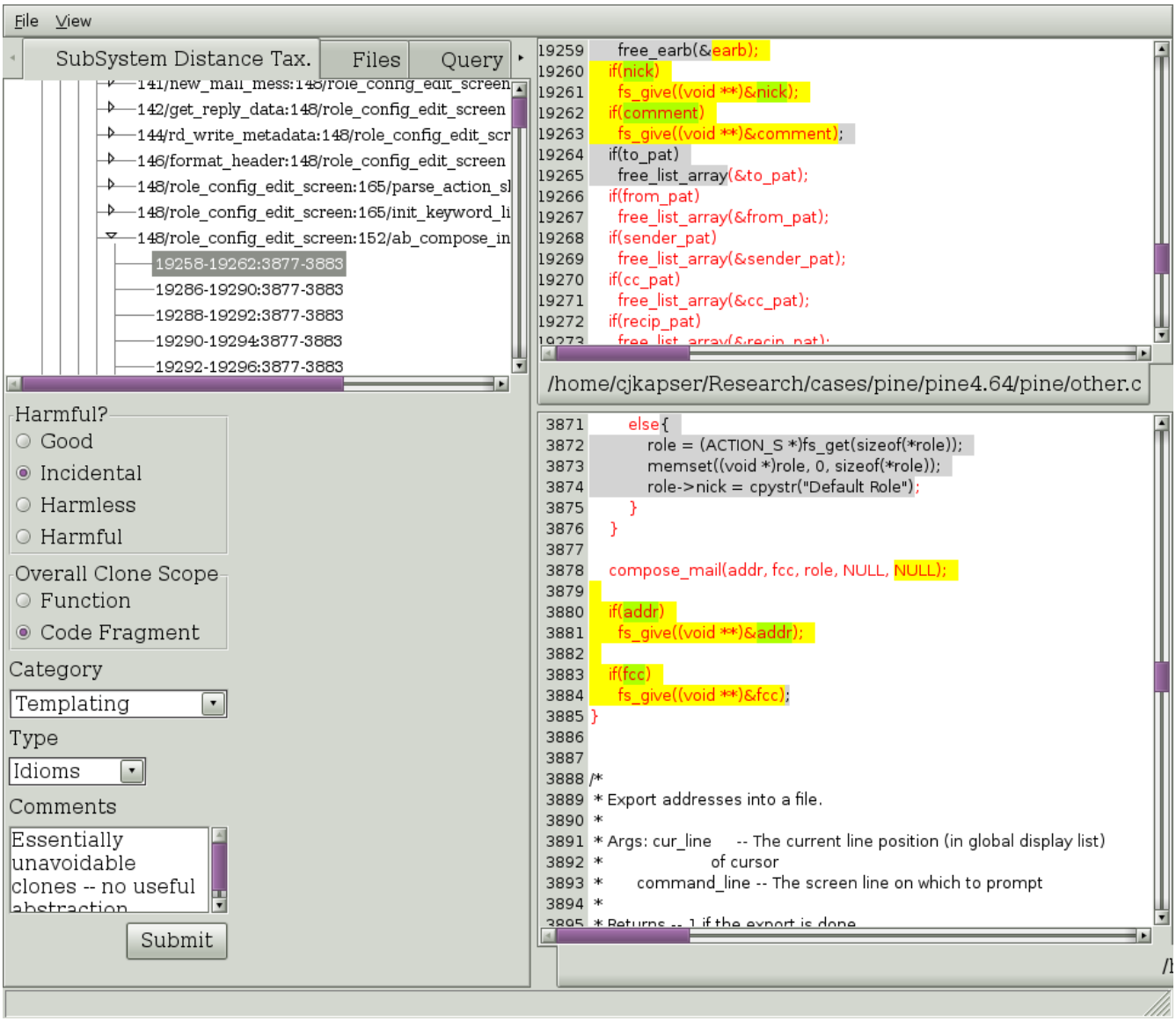

Figure 3: A clone selection in the subsystem distance taxonomy.

The default tab is the Subsystem distance taxonomy tab. This tab allows you to browse clones according to an automatic classification, described in [1]. The current implementation is not complete (in the middle of refactoring). See the tool for current level of implementation. To navigate clones in this tree select nodes in the tree. Each node represents a level in the taxonomy, indicated by the name of the node in the tree, until you reach a set of nodes labeled with the format “integer/string:integer/string” (ex. 148/role_config:152/ab_compose). This indicates you have hit the end of the categorization for this sub-tree, and the children are now Regional Group of Clones (RGCs). This is a set of clones that comprise the cloning between two regions. In Figure 3, on the left we see a RGC that has been expanded and one of the clones has been selected. On the lower left you can that the type of cloning this RGC contains has been filled in. The basis for the categories, harmfulness, and scope is documented in [2]. To commit annotations to the database, simply fill them in and select “Submit”.

On the right, the selected region is show in red text, and the selected clone is shown in yellow and green highlights. The highlighting provides an indication of overall clone similarity by illustrating the simple diff of the tokens of the two clones. Yellow highlighting indicates the tokens in the clones are the same,

green indicates the tokens are different. This highlighting can be applied to the entire region if you wish to see the longest common subsequence between the regions. Simply right click on the TEXT of one of the regions and select “Diff regions” from the menu that will pop-up. You will also notice in Figure 3 that some of the text is highlighted in gray with black text. This indicates there are clones between the two files that are not part of this RGC. This is intended to provide information about the degree of cloning between the two files: scrolling through the two files will show more highlighted text. The reader should also node that each level of the tree is annotated by an integer “(integer)”. This integer indicates the number of clones contained in the subtree of each node.

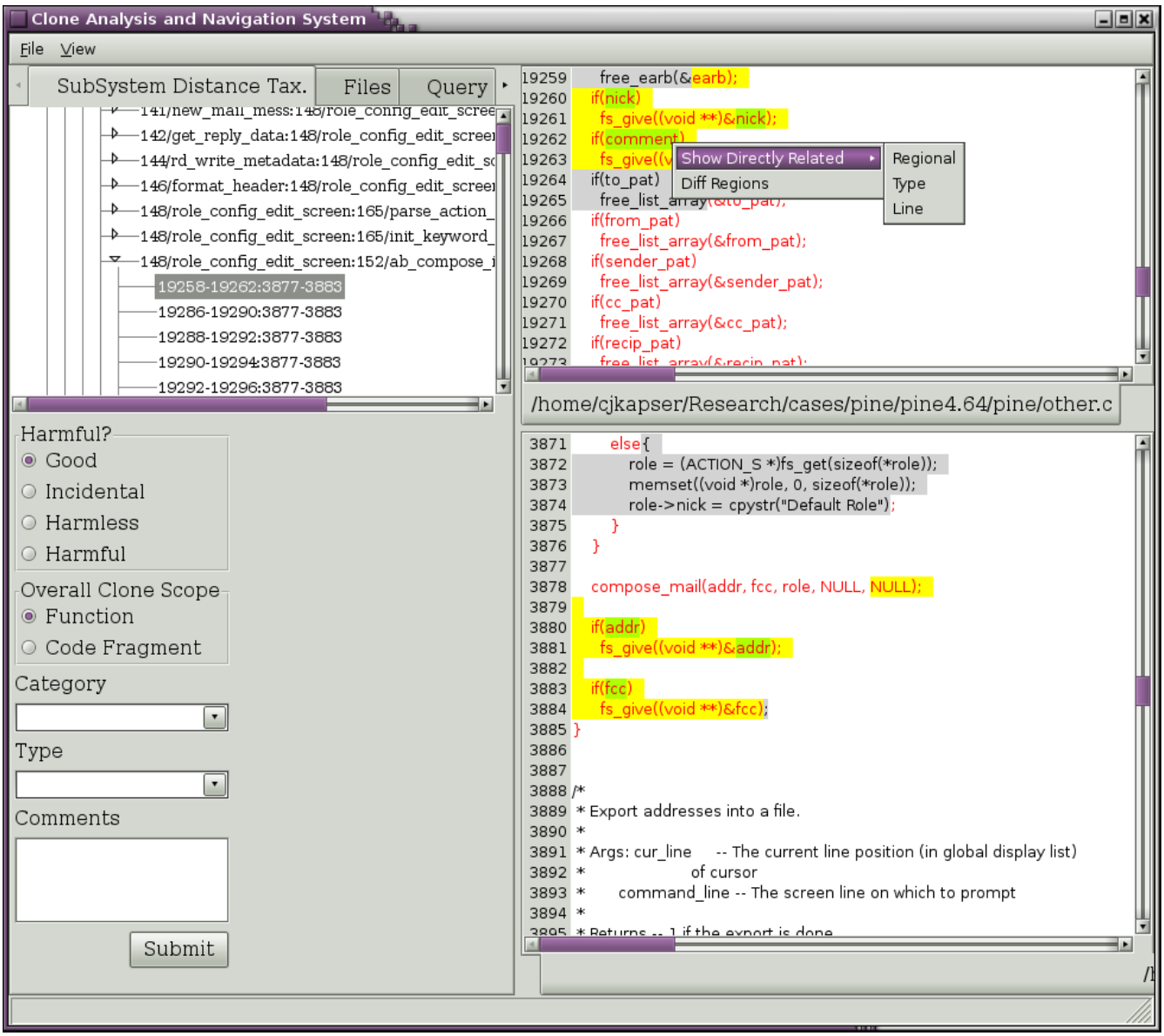

Figure 4: Query for related clones.

If you wish to see clones that are related to RGC you are currently viewing you can ask to see all clones that occur in one of the two regions. Right click on the text of interest and select one of the sub-menu items under the “*Show Directly Related*” pop-up menu item (seen in Figure 4). The three choices are regional, type, and line. Selecting “Regional” will return all clones that have one segment in the selected region. Selecting “Type” will return all clones that have one segment in the selected region and

are of the same type you are currently viewing. Selecting “Line” will return all clones that have one segment covering the line you right-clicked on. In the past you could also make similar queries about clones using transitive closure, this feature is currently being reimplemented. Figure 5 depicts the results of selecting “Line” in Figure 4. The results, organized by the same taxonomy used in the first tab, are shown in the Query tab. This tree has the same behavior as described for our first tab.

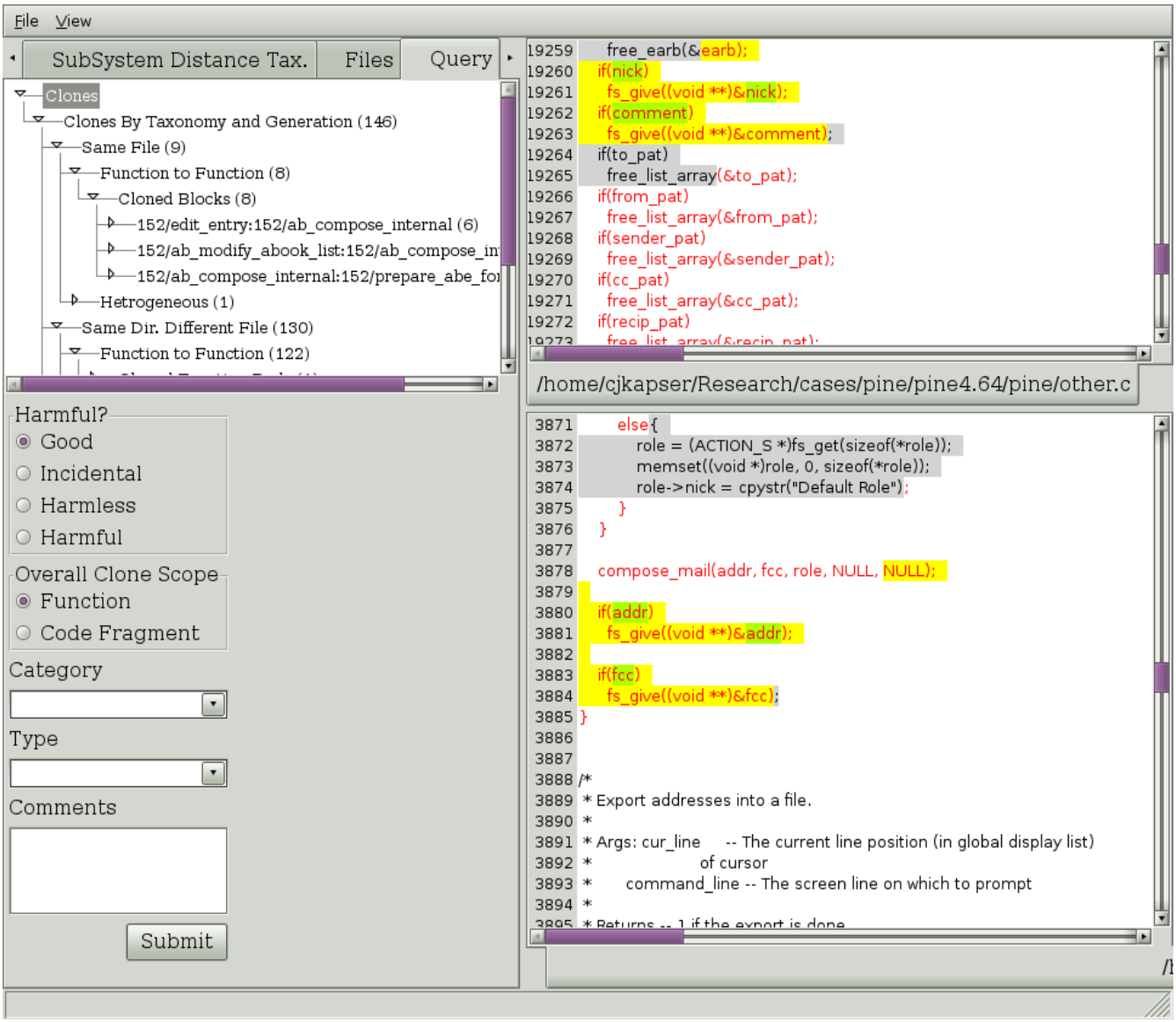

Figure 5: Result of Directly Related -> Line query.

The clone detection method used by CLICS often contains false positives. These can be removed from the database as you discover them. Simply select the clone that is a false positive and then right click on it. A pop-up menu will appear (Figure 6). To remove all clones in the clone group, select “Remove Group of Clones”. To remove an individual clones, select “Remove Clone”. *Bug: Clones are not being recategorized after removing an individual clone. To be fixed.*

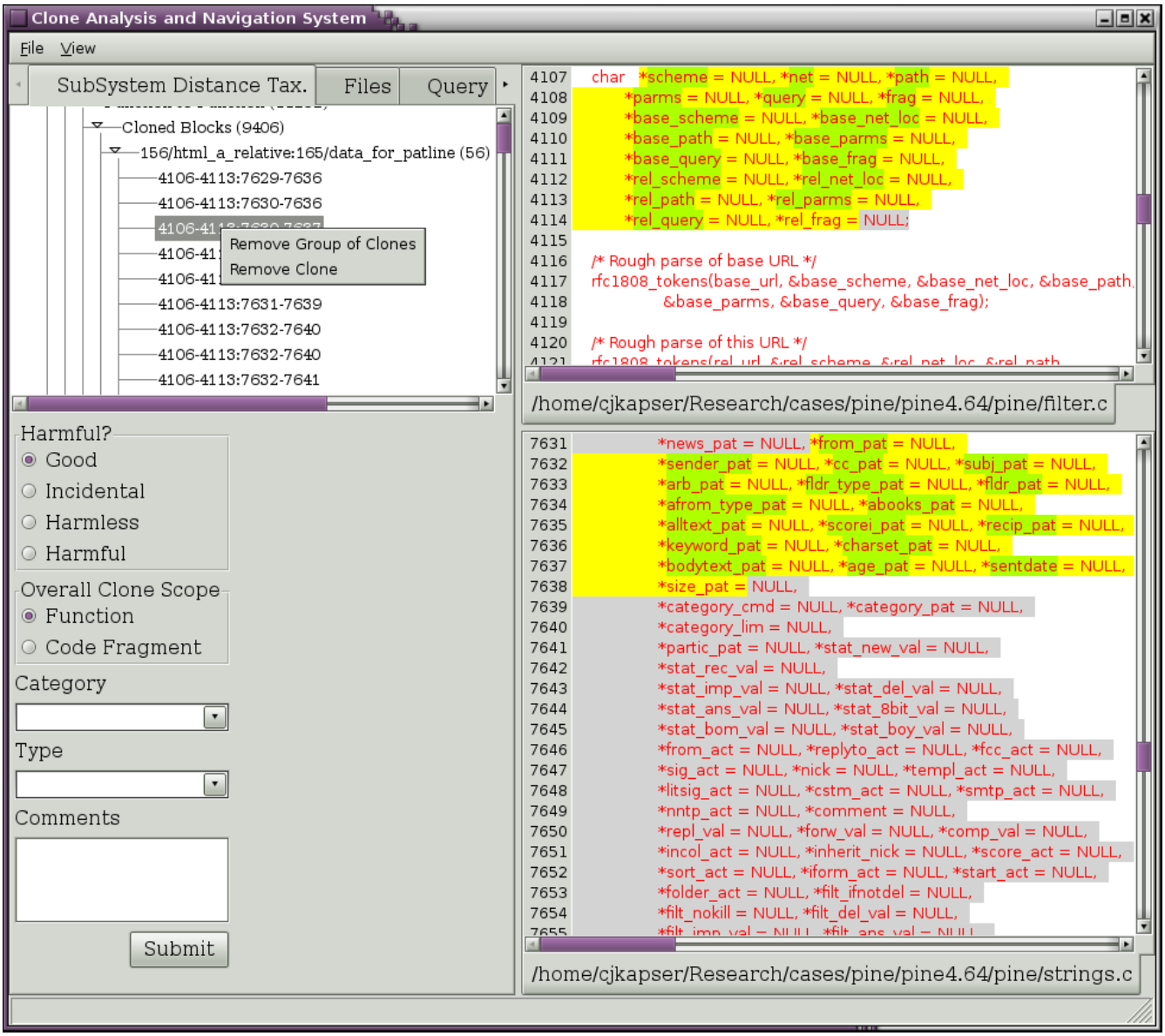

Figure 6: Menu to remove clones.

Browsing the clones found in the taxonomy can be a good way to get acquainted with the types of cloning that occur within a software system, but an investigation of the clones within the context the software organization is necessary if one is to understand how clones have been used. There are two mechanisms for exploring clones in relationship to the source code organization. The first is the "*Files*" tab (Figure 7). The "*Files*" tab provides several useful statistics about cloning within and going out of software entities (sub-systems, files, and regions). Figure 7 shows the cloning statistics for the top level source directory pine4.64. On the left is a tree representing the source hierarchy of pine. On the right we can see the grid of statistics for pine4.64 (selected on the left). The grid on the left can be sorted according to any column (done so by selecting the column header). Using the source tree and the stats grid, the user can find cloning of interest (example: Figure 7 shows that 46.8% of the clones in pine occur in the imap subsystem while imap only contains 28.6% of the lines of code).

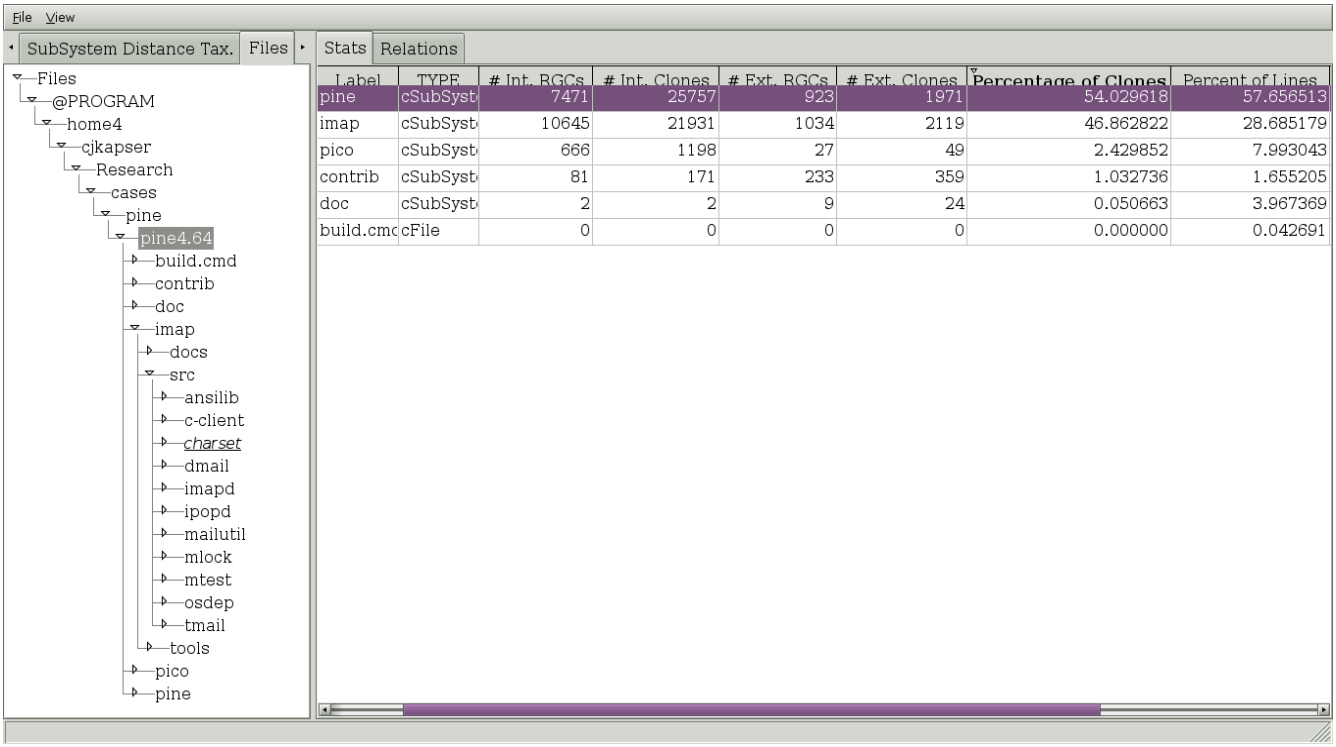

Figure 7: Statistics of cloning by software entity.

In Figure 7 the grid has been sorted by “Percentage of Clones”. The stats grid contains the following columns:

1. Label: the name of the system entity.
2. Type: Defines the system entity type. Can be cSubSystem, cFile, cFunction, or cObject. cSubSystem indicates the system entity is a subsystem in the source code. cFile indicates the system entity is a source file. cObject indicates the system entity is a data type definition in the source code (such as a struct, union, enumerator, etc) and cFunction indicates the system entity is a function or procedure.
3. # Int. RGCs: The number of RGCs with both regions occurring within the system entity (internal RGCs).
4. # Int. Clones: The number of clones with both segments occurring within the system entity (internal clones).
5. # Ext. RGCs: The number of RGCs with only one region occurring within the system entity (external RGCs).
6. # Ext. Clones: The number of clones with one segment within the entity and the other segment occurring outside of the entity (external clones).
7. Percentage of Clones: The percentage of clones with at least one segment occurring within the system entity.
8. Percent of Lines: The percentage of lines of code that occur within the system entity.
9. Total Lines Cloned: The number of lines cloned (currently disabled).

Using the above information you can find areas with a high density of clones (such as when a disproportionate number of clones occur in a relatively small number of lines of code), possible false positives (such as occurs in relatively simple code, indicated by a large number of clones by few RGCs), and cloned subsystems due to forking (indicated by high numbers of external clones). If you find a

system entity of interest and wish to see the clones within a system entity, right click on the system entity in the file tree, and select one of the menu items in the pop-up menu (Figure 8). The menu items “Clones within Entity” and “Clones Going Out of Entity” will display the internal clones and external clones of the system entity in the Query tab. “Show Internal Summary” provides a summary of internal cloning of the selected system entity. “Remove Internal Clones” will remove all clones (permanently) from the analysis.

If you find a system entity that you wish to remove from the analysis (perhaps it has so many clones it skews the results) you can exclude it from the analysis. “Exclude Entity” will exclude the entity from the analysis temporarily (the entity can be included using a similar pop-up menu). This action will only be applied after first selecting “Exclude Entity” and then applying the change using “Apply Changes”. After excluding the entity, it will appear italicized and underlined as “charset” appears in Figure 8.

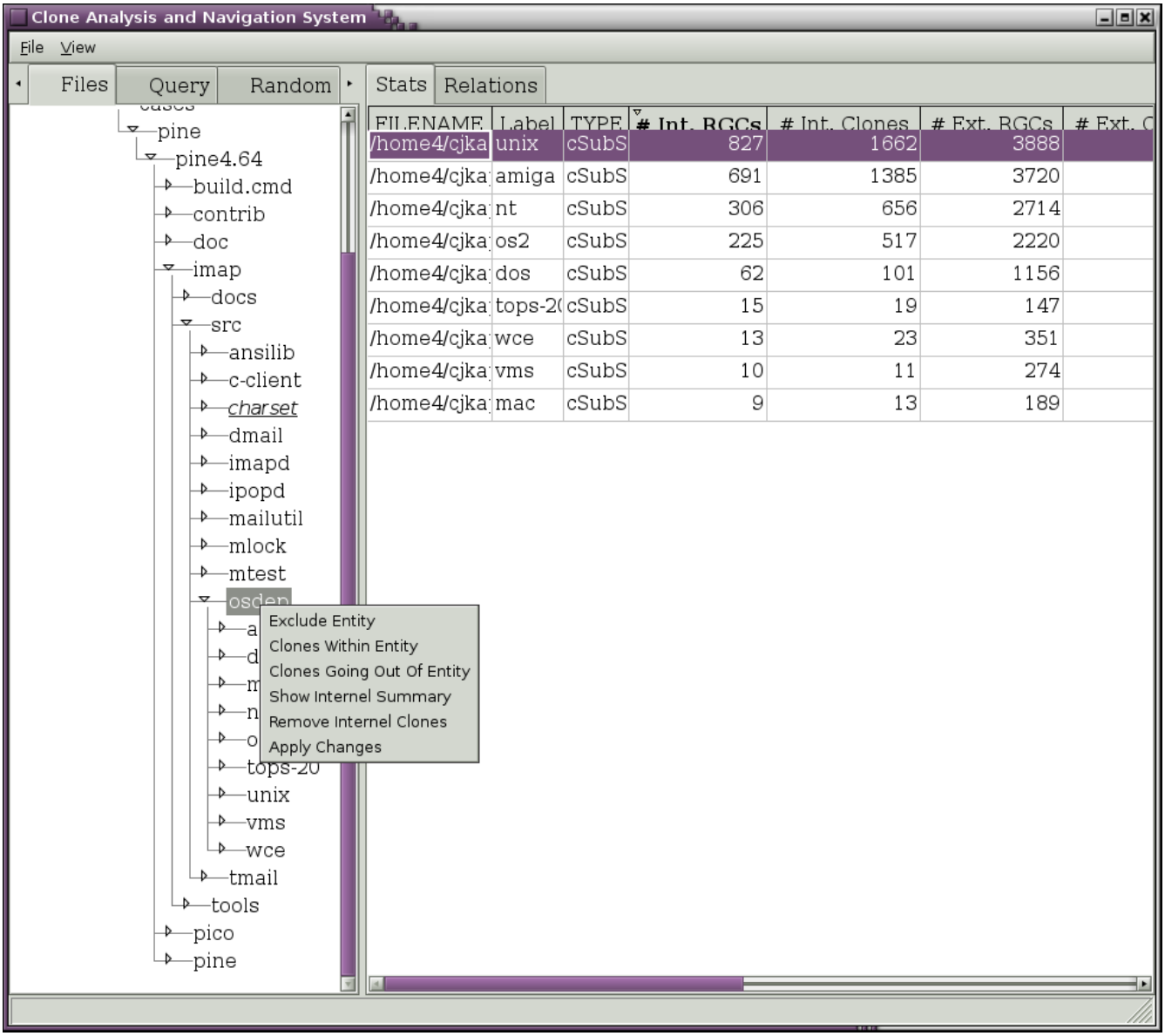

Figure 8: System entity pop-up menu.

Information about the external clones (clones going out of a system entity) can be found in the "*Relations*" tab in the right hand side of the Files view (shown in Figure 9). The Relations tab enables the user to walk through the another file tree to gain information about the degree of cloning between the system entity selected in the left tree and the system entity selected in the left tree. Figure 10 shows that imap and contrib share 290 clones and imap and pine share 1829 clones. To view the clones comprising this relationship, right click on the system entity of interest and select "Show Relation" from the pop-up menu (Figure 10), the results will be shown in the Query tab.

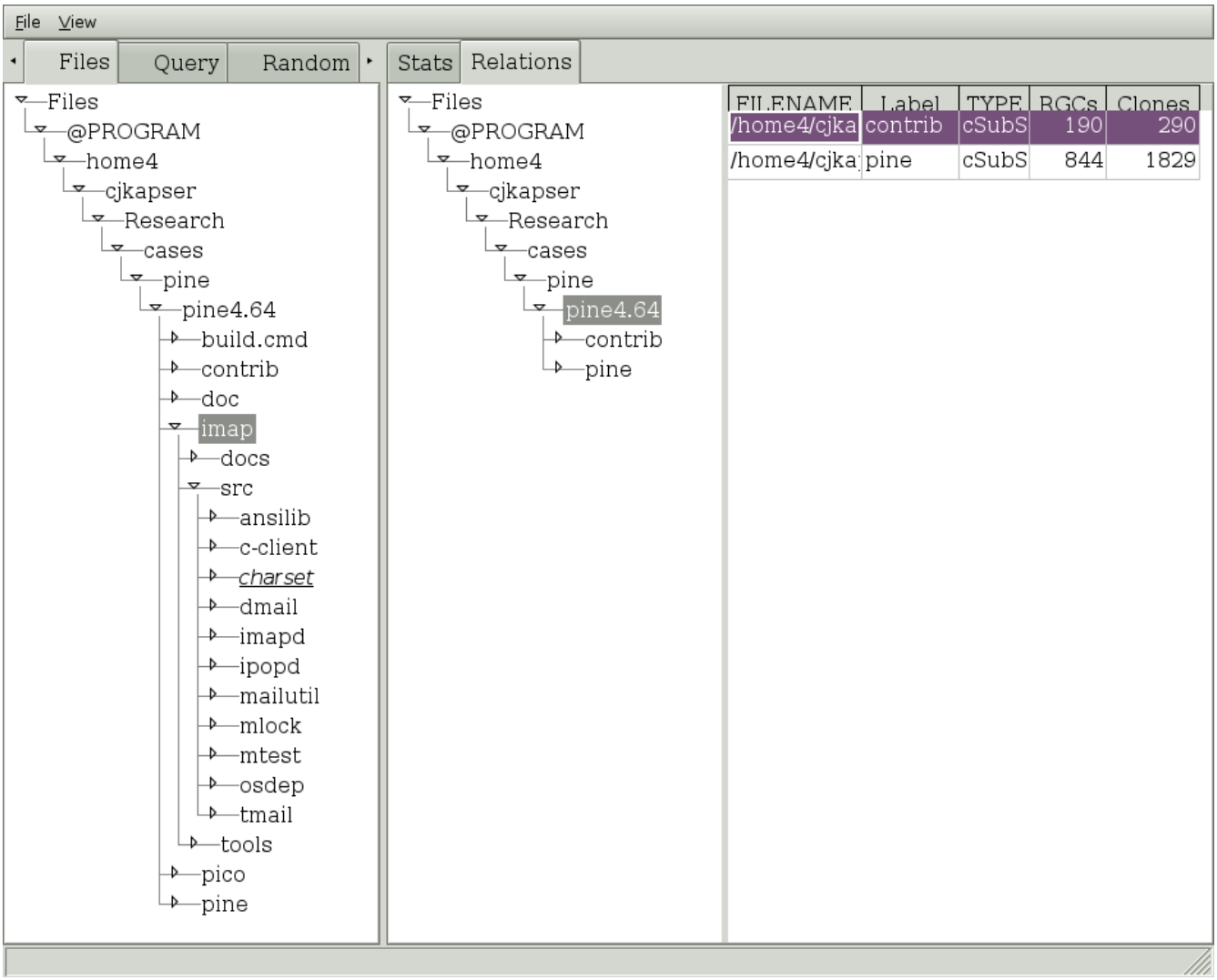

Figure 9: Relations tab.

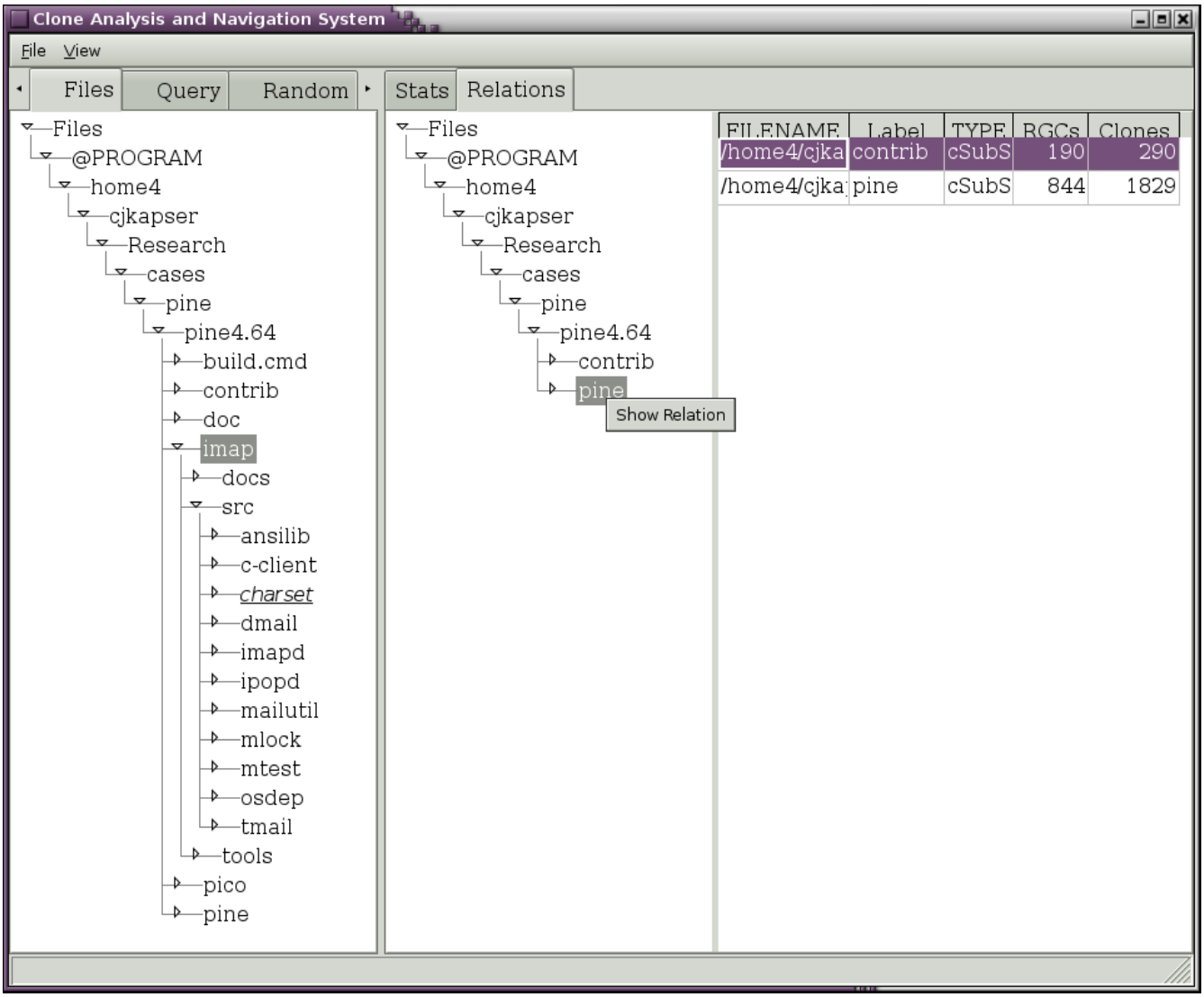

Figure 10: Show relation pop-up in relations tab.

To document:

- summary panel
- lsedit view

Appendix 1. ExtractRegions.py

ExtractRegions.py generates a list of code regions extracted from a set of input files. It uses a modified version of ctags to extract this information.

Extract Regions File Format:
There are two sections to the output of ExtractRegions.py:

1. Files, and
2. Regions.

Each section is delimited by begin{} and end{} statements. Ex. The Files section looks something like:

```
begin{files}
0       17      /home/cjkapser/Research/cases/pine4.64/pico/buffer.c
1       12      /home/cjkapser/Research/cases/pine4.64/pico/edef.h
2       2       /home/cjkapser/Research/cases/pine4.64/pico/resource.h
3       26      /home/cjkapser/Research/cases/pine4.64/pico/word.c
4       3       /home/cjkapser/Research/cases/pine4.64/pico/window.c
5       69      /home/cjkapser/Research/cases/pine4.64/pico/browse.c
6       34      /home/cjkapser/Research/cases/pine4.64/pico/msdlg.c
7       49      /home/cjkapser/Research/cases/pine4.64/pico/basic.c
8       43      /home/cjkapser/Research/cases/pine4.64/pico/pico.h
9       6       /home/cjkapser/Research/cases/pine4.64/pico/efunc.h
end{files}
```

and the regions section looks like:

```
begin{regions}
v       1       1       rcsid           0       0
m       2       38      MiscRegion      0     1       0
p       39      41      sgetlin         2       0
m       42      54      MiscRegion      1     3       0
f       55      66      anycb           4       0       56
m       68      76      MiscRegion      2     5       0
f       77      137     bfind           6       0       80
m       139     148     MiscRegion      3     7       0
f       149     178     bclear          8       0       151
m       181     187     MiscRegion      4     9       0
f       188     264     packbuf         10      0       192
m       267     270     MiscRegion      5     11      0
f       271     323     readbuf         12      0       273
m       326     330     MiscRegion      6     13      0
f       331     366     sgetline       16      0       336
end{regions}
```

The files section.

Each line of the files section contains three sections:

- file id,
- number of lines in the file, and
- file path.

The file id is a unique numeric identifier, used to uniquely identify and cross reference each file in the regions section.
The number of lines in the file represents the number of lines in the text file (all lines, including comments, white space, etc).

The file path is the path used by the extractor to read the file.

The regions section.
Each line in the regions section contains the following fields:

1. the region type,
2. the starting line of the region,
3. the ending line of the region,
4. the name of the region,
5. the region id, and
6. the file id of the region.

Regions representing methods, procedures, and functions include an additional field denoting the line number of the opening brace.

Region types are taken directly from the ctags output. They are a single character. For C/C++/Java, the following regions types are used:

- g: enumerator name
- d: macro
- n: namespace
- c: class
- f: function
- p: prototype definition
- s: struct
- t: type definition
- u: union
- e: enumerator
- x: extern
- v: variable definition

In addition to the ctags region types, the extraction script adds a misc region to ensure all lines of the file are covered:

- m: miscellaneous region.

Appendix 2. clics_clone_detection

Using a suffix tree based string searching/matching algorithm, the clics_clone_detection program provides a list of common substrings found in a set of input files.

There are 2 sections in the output file:

1. the files list, and

2. the clone pair list.

The output format was originally very similar to the CCFinder output, but has evolved a little over time. The sections are delimited by start{} and end{} lines. For example, the files section looks like:

```
start{files}
0       367     /home/cjkapser/Research/cases/pine4.64/pico/buffer.c
1       181     /home/cjkapser/Research/cases/pine4.64/pico/edef.h
2       134     /home/cjkapser/Research/cases/pine4.64/pico/resource.h
3       644     /home/cjkapser/Research/cases/pine4.64/pico/word.c
4       60      /home/cjkapser/Research/cases/pine4.64/pico/window.c
5       2586    /home/cjkapser/Research/cases/pine4.64/pico/browse.c
6       1003    /home/cjkapser/Research/cases/pine4.64/pico/msdlg.c
7       878     /home/cjkapser/Research/cases/pine4.64/pico/basic.c
8       605     /home/cjkapser/Research/cases/pine4.64/pico/pico.h
9       342     /home/cjkapser/Research/cases/pine4.64/pico/efunc.h
10      188     /home/cjkapser/Research/cases/pine4.64/pico/osdep/os-mnt.h
11      173     /home/cjkapser/Research/cases/pine4.64/pico/osdep/os-gen.h
12      179     /home/cjkapser/Research/cases/pine4.64/pico/osdep/os-nto.h
13      174     /home/cjkapser/Research/cases/pine4.64/pico/osdep/os-gsu.h
14      173     /home/cjkapser/Research/cases/pine4.64/pico/osdep/os-mct.h
end{files}
start{clones}
3       166,0,4922      184,36,5546     3       200,0,6063      218,36,6687  \
115     0.026087        0.076923
3       166,0,4922      180,0,5333      3       235,0,7267      249,0,7673 \
77      0.012987        0.040000
3       181,22,5374     189,47,5817     3       250,19,7711     258,47,8079 \
80      0.000000        0.000000
3       200,0,6063      214,0,6474      3       235,0,7267      249,0,7673 \
77      0.000000        0.000000
3       432,21,12880    453,0,13396     3       508,34,14994    524,0,15357 \
60      0.083333        0.227273
5       492,18,14069    513,0,14553     5       514,16,14571    535,0,15071 \
141     0.035461        0.096154
5       503,34,14344    511,0,14534     5       1597,18,39342   1605,0,39544 \
61      0.016393        0.041667
5       503,38,14348    515,19,14610    5       630,15,17328    644,19,17574 \
72      0.027778        0.076923
5       504,28,14378    517,12,14651    5       598,23,16542    611,17,16806 \
69      0.072464        0.200000
5       504,28,14378    515,19,14610    5       661,23,17951    672,19,18158 \
63      0.047619        0.130435
end{clones}
```

* Note, in the above example the \ indicates a line wrap.

The files section.

Each line in the files section contains 3 fields:

1. file id,
2. file length in lines, and
3. file path.

File id is a unique numeric identifier used to reference a file in the clones section.

File length in lines indicates the number of lines in the input file (including empty lines and documentation).

File path indicates the path used to read the file.

The clones section.

Each line in the clones section describes a single clone pair. For the clics_clone_detector a clone pair is composed of two segments of code. Each line can be decomposed into three sections:

1. Segment 1,
2. Segment 2, and
3. clone summary.

```
3  166,0,4922  184,36,5546  3  200,0,6063  218,36,6687  115  0.026  0.076
Segment 1                  Segment 2                  Clone summary
```

Each segment contains the following fields:

1. file id – the file the segment occurs in;
2. segment start – the start line, start column, and start character;
3. segment end – the end line, end column, and end character.

The clone summary includes:

1. clone length - the length of the clone, in tokens,
2. total difference - ratio of differences to clone length between the segments using all tokens as a comparison,
3. identifier difference - ratio of differences to number of identifiers in the segments using only identifiers in the comparison.

References:

(1) "Improved Tool Support for the Investigation of Duplication in Software", by Cory Kapser and Michael W. Godfrey. Proc. of the 2005 Intl. Conference on Software Maintenance (ICSM-05), Budapest, Hungary, 25-30 Sept 2005.

(2) “`Cloning Considered Harmful' Considered Harmful: Patterns of Cloning in Software", Cory J. Kapser and Michael W. Godfrey. Extended version of WCRE-06 Best Paper. Accepted to appear in Empirical Software Engineering (Springer).