Heptane: hands-on session

1 Installation

For the tutorial a virtualbox[1] machine[2] is available at this url:
http://www.irisa.fr/alf/downloads/hardy/tutor/Heptane-VM.ova

To install it, you have to import (File > Import Appliance...) the virtual machine and you have to generate a new mac address during the import. Then once started, you may need to change the keyboard layout. It is in the menu (top left corner) > Settings > Keyboard > Layout. In case the screen does not scale, you have to install the guest additions (Device > Insert Guest Additions CD image). If needed the root password is just: password

2 First execution

Heptane is already installed in the directory `Desktop/Heptane`. You just have to open a shell and type the following commands:

```
cd Desktop/Heptane
./analysis.sh matmult
```

The output should provides something similar to Figure 1.

3 The toolchain

In this part, you will go through the toolchain from extracting a program to estimating its WCET.

3.1 HeptaneExtract

The first thing to do is to go into the directory `benchmarks`. For now, we will focus on the tutor benchmark. In the directory `tutor`, you have only the c file of the benchmark. As you can see, it is a very basic program for now. To extract it, you have to be in directory `benchmarks` and type the following command:

```
./extract.sh tutor
```

This script generates the configuration file for the extraction and executes HeptaneExtract. Once executed you have several files in the tutor directory:

[1] virtualbox: https://www.virtualbox.org

[2] The vm is based on xubuntu. Basic softwares are already installed for the tutorial (Kate editor, Document viewer (pdf), evince... In case you want to install your favorite software, you can use the *apt-get install application* command.

```
user@Hepane-VM:~/Desktop/Heptane$ ./analysis.sh matmult
Reading configuration file
Executing from configuration file
[INFO]  ICacheAnalysis: MUST done: 0.007825
[INFO]  ICacheAnalysis: PS done: 0.00223
[INFO]  ICacheAnalysis: MAY done: 0.010891
[INFO]  ICacheAnalysis: MUST done: 0.021785
[INFO]  ICacheAnalysis: PS done: 0.003603
[INFO]  ICacheAnalysis: MAY done: 0.01382
[INFO]  DcacheAnalysis: MUST done: 0.004449
[INFO]  DcacheAnalysis: PS done: 0.007091
[INFO]  DcacheAnalysis: MAY done: 0.006934
[INFO]  DcacheAnalysis: MUST done: 0.016503
[INFO]  DcacheAnalysis: PS done: 0.006675
[INFO]  DcacheAnalysis: MAY done: 0.019354
WCET: 1795580
L1      type ICACHE     references 426086       hits 426054     misses 32
L1      type DCACHE     references 28929        hits 4924       misses 24005
L2      type ICACHE     references 32   hits 3  misses 29
L2      type DCACHE     references 24005        hits 23922      misses 83
WCET: 1795580
```

Figure 1: Heptane output

- annot.xml represents the loops maxiter information
- configExtract.xml is the configuration file used for the extraction
- tutor.exe is the binary file
- tutor.objdump is the result of the objdump command on the binary
- tutor.readelf is the result of the readelf command on the binary
- tutor.xml is the CFG used by HeptaneAnalysis

You can open tutor.xml to have a look at the representation of the program.

3.2 HeptaneAnalysis

For the WCET analysis of tutor you have to go into the directory `Heptane` and then to execute the following command:

```
./analysis.sh tutor
```

This script generates the configuration file for the WCET analysis and executes Heptane-Analysis. Once executed you can observe the WCET and the cache statistics in the shell. You can also observe the different files created in the `tutor` directory.

- tutor.pdf is a graphical representation of the CFG
- res[AnalysisName].xml is the output xml after the analysis where attributes have been added

A `configWCET.xml` file has been created in the directory `Heptane`. This is the configuration file used for the WCET analysis.
Open it and remove the comment[3] surrounding the CODELINE and HTMLPRINT analyses. You can analyze tutor again but with these two analyses activated and by typing the following command:

```
./bin/HeptaneAnalysis configWCET.xml
```

This command replaces the script `analysis.sh` since the configuration file is already created. Once executed you can open the file tutor.html generated in the `tutor` directory. You can see for each line of code the execution frequency along the worst-case execution path. They are all executed once since the benchmark is very simple with only one execution path.

3.3 A bigger benchmark

You can now create your own benchmark directly by changing the `tutor.c` file or you can choose one of the existing benchmarks[4].

When it is done, extract it and analyze it by following the description of the previous sections. You can spend some time to look at the different files generated during the extraction and the WCET analysis.

3.4 Analysis parameters: configWCET.xml

Until now, you have just used the default parameters. You can change them. For instance the architecture parameters with the replacement policy of instruction caches or the cache size to see the impact on the WCET. For the each analysis you can:

- deactivate some part of the cache analysis (turn off the persistence analysis for instance)
- change the method of the IPET analysis (METHOD_CACHE_PIPELINE for instance)
- change the printing information of the simpleprint analysis.
- ...

4 Your own analysis

In this part you will develop an analysis to provide the ratio of load and store instructions in a program.

4.1 Dummy analysis

To avoid the parsing of the configuration file, you will use the dummy analysis already defined in the directory `Heptane/src/HeptaneAnalysis/src/Specific/DummyAnalysis`.

To apply this analysis, you just have to insert the following XML line in the WCET configuration file:

```
<DUMMYANALYSIS keepresults="true" input_file ="" output_file ="" />
```

To compile it, you just have to call the command `make` in the directory `Heptane/src/`.

[3] comments in XML start with <!- - and end with - ->
[4] The Mälardalen benchmarks come from http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

4.2 Different steps

1. Counting the number of static instructions
2. Counting the number of static load/store instructions
3. Extension: calling context (see the presentation)

4.3 Some indications

- The different counters needed can be declared as attributes in the dummy analysis class
- The function `applyToAllNodesRecursive` can be used (see the presentation). This allows to focus on the analysis of a node. This function can be used as follows:

```
//computeNBInstruction definition
static bool computeNBInstructionCTX (Cfg * c, Node * n, void *param){
    DummyAnalysis * da = (DummyAnalysis *) param;
    ...
    return true;
}
// In the PerformAnalysis function of the dummy analysis class
AnalysisHelper::applyToAllNodesRecursive (p, computeNBInstruction, this);
```

- The documentation of the CFG library is available here: /home/user/Desktop/Heptane/src/cfglib_install/doc/generated-doc/html/index.html

4.4 Useful functions (not in the presentation)

- For the Instruction
  - `std::string GetCode()` returns the assembly code line
  - `Arch::isLoad(const string& instr)` returns true if instr is a load (requires `#include "arch.h"`)
  - `Arch::isStore(const string& instr)` returns true if instr is a store (requires `#include "arch.h"`)
- For the Node
  - `std::vector<Instruction*> GetInstructions ()` returns the instruction vector