wcetac : A WCET Analyzer/Checker – User Manual

Vitor Rodrigues
Rochester Institute of Technology (RIT)

May 15, 2014

This document gives the basics for installing and use the Haskell prototype of a WCET analyze/checker called ”wcetac”.

1 System Requirements

The system requirements are the GHC compiler and interpreter (http://www.haskell.org/ghc), the Cabal package management system (the Common Architecture for Building Applications and Libraries), the GLPK (GNU Linear Programming Kit) package, and the command-line tool ”cabal-install”.

1. The Cabal package is usually installed together with GHC compiler because it is included in the Haskell Platform (http://www.haskell.org/platform/).
2. The GLPK package (version 4.48) can be installed:
   (a) in Linux machines using the appropriate packaging systems: ”yum” for Fedora and alike and ”apt-get” for Debian-based distributions like Ubuntu. For example, in Fedora, execute ”yum install glpk glpk-devel”.
   (b) in Mac OS X machines, the easiest way is to install the ”macports” tool (http://www.macports.org/install.php). Then, execute ”sudo port -v selfupdate”, followed by ”sudo port install glpk”.
3. Similarly, the command-line tool ”cabal-install” (version 1.16.0.2) can be installed:

(a) in Linux machines by executing "yum install cabal-install"

(b) in Mac OS X machines by executing "sudo port install hs-cabal-install"

After installing "cabal-install", execute "cabal update". This will allow automatic installation of dependencies when installing "wcetac". The following message should pop-up, but it should be ignored. The prototype installation have been tested for version 1.16.0 of the Cabal library only.

```
Note: there is a new version of cabal-install available.
To upgrade, run: cabal install cabal-install
```

2 Installation of the "wcetac" tool

1. Download the "wcetac-10.0.tar.gz" file [here]. A directory "wcetac-1.0.0" will be create that include all the source files, the C benchmarks and the corresponding assembler files (extension .s). The inclusion of these files avoid the need to cross-compile GCC to the ARM platform in your machine. The package description file "wcetac.cabal" is also included.

2. To install the "wcetac" tool (including all Haskell dependencies), simply execute "cabal install wcetac.cabal" inside the directory "wcetac-1.0.0". This is put the binary "wcetac" under the directory "$(HOME)/.cabal/bin" and the assembler files under "$(HOME)/.cabal/share/wcetac-1.0.0".

In order to check if the dependencies were well installed, you can execute "ghc-pkg list". The list of package that were installed in your local GHC package repository will show after the line "$(HOME)/.ghc/(...)/package.conf.d" with the following:

```
HUnit-1.2.5.2
MissingH-1.2.1.0
Vec-1.0.1
cmdargs-0.10.7
hslogger-1.2.4
logfloat-0.12.1
regex-base-0.93.2
regex-compat-0.95.1
regex-posix-0.95.2
xml-1.3.13
```

3 Run the "wcetac" tool

To the run the WCET analyzer/checker the easiest way to change directory into "$HOME/.cabal/bin" and first execute "./wcetac –help". The output will be:

```
WCETAC v1.0.0, Vitor Rodrigues

wcetac [OPTIONS] [FILE]
  WCET Static Analyzer/Checker

Common flags:
  -g --graph[=FILE]   export graph to <file>
  -d --datadir[=DIR]  benchmarks location <datadir>/
  -t --tdm            use the TDMA composable arbiter
  -l --lr             use the LR-rate server abstraction
  -a --analyze        run the WCET analyzer
  -c --check          run the WCET checker
  -n --noreduce       don't apply the sequential recursive transformations
  -? --help           Display help message
  -V --version        Print version information
  -v --verbose        Loud verbosity
  -q --quiet          Quiet verbosity

To analyze the WCET of an assembler <file> (without extension) use [FILE]
To check the WCET estimate use --check or -c
To use the TDM arbiter instead of the default LR-server use --tdma or -t
If the 'wcetac' is inside PATH, the default location of data
'~/.cabal/share/wcetac-1.0.0/' can be changed using --datadir or -d

Example of how to 'analyze' and 'check' the WCET of the assembler file
'benchmark.s': 'cd ~/.cabal/bin' and then invoke './wcetac --check benchmark'
```

In order to analyze and check the WCET of a particular benchmark, for example `fibcall` in verbose mode (this will print all the information that is presented in the paper, such as verification times, certificate sizes, etc.), and using the latency-rate abstraction (default), you can simply execute:

./wcetac -v --check fibcall

4 Visualization of Dependency Graphs

The dependency graphs extracted from the assembler files can be viewed using the visual editor yEd (http://www.yworks.com). The default filename is "out.graphml", but a different name can be given with the option "--graph=[path to file]".

By default, the dependency graphs are transformed according to the *sequential* and *recursive* algebraic rules described in the paper. In order to disable these transformations use the option ”--noreduce”, or simply ”-n”.

As an example, consider the following C code with two nested loops:

```
int main(void)
{
  int i;
  for (i = 1; i <= 5; i++)
  {
    int x = 4;
    while (x>0)
    {
       x--;
    }
  }
  return 0;
}
```

The corresponding assembler and the labeled relation semantics is:

```
                  n1 <--|  mov   ip, sp                <--| r0 {"main"}; 1       [d0]
                  n2 <--|  stmfd sp!, {fp,ip,lr,pc}    <--| n1                   [d1]
                  n3 <--|  sub   fp, ip, #4            <--| n2                   [d2]
                  n4 <--|  sub   sp, sp, #8            <--| n3                   [d3]
                  n5 <--|  mov   r3, #1                <--| n4                   [d4]
                  n6 <--|  str   r3, [fp, #-20]        <--| n5                   [d5]
                 n19 <--|  b     52                    <--| n6                   [d6]
                  n8 <--|  mov   r3, #4                <--| n7 {(".L3","main")}  [d7]
                  n9 <--|  str   r3, [fp, #-16]        <--| n8                   [d8]
                 n13 <--|  b     16                    <--| n9                   [d9]
                 n11 <--|  ldr   r3, [fp, #-16]        <--| n10 {(".L5","main")} [d10]
                 n12 <--|  sub   r3, r3, #1            <--| n11                  [d11]
                 n13 <--|  str   r3, [fp, #-16]        <--| n12                  [d12]
                 n14 <--|  ldr   r3, [fp, #-16]        <--| n13 {(".L4","main")} [d13]
                 n15 <--|  cmp   r3, #0                <--| n14                  [d14]
                 n10 <--|  bgt   -20                   <--| H15                  [d15]
                 n16 <--|  bgt   -20                   <--| n15                  [d16]
                 n17 <--|  ldr   r3, [fp, #-20]        <--| n16                  [d17]
                 n18 <--|  add   r3, r3, #1            <--| n17                  [d18]
                 n19 <--|  str   r3, [fp, #-20]        <--| n18                  [d19]
                 n20 <--|  ldr   r3, [fp, #-20]        <--| n19 {(".L2","main")} [d20]
                 n21 <--|  cmp   r3, #5                <--| n20                  [d21]
                  n7 <--|  ble   -56                   <--| H21                  [d22]
                 n22 <--|  ble   -56                   <--| n21                  [d23]
                 n23 <--|  mov   r3, #0                <--| n22                  [d24]
                 n24 <--|  mov   r0, r3                <--| n23                  [d25]
                 n25 <--|  sub   sp, fp, #12           <--| n24                  [d26]
exit  {(".L2","main")} <--|  ldmfd sp, {fp,sp,pc}      <--| n25                  [d27]
```

After running the tool, open the (default) ”out.graphml” file inside yEd, and in the menu item ”Layout” choose ”Hierarchical”. The dependency graph in Figure 1 should be displayed as:

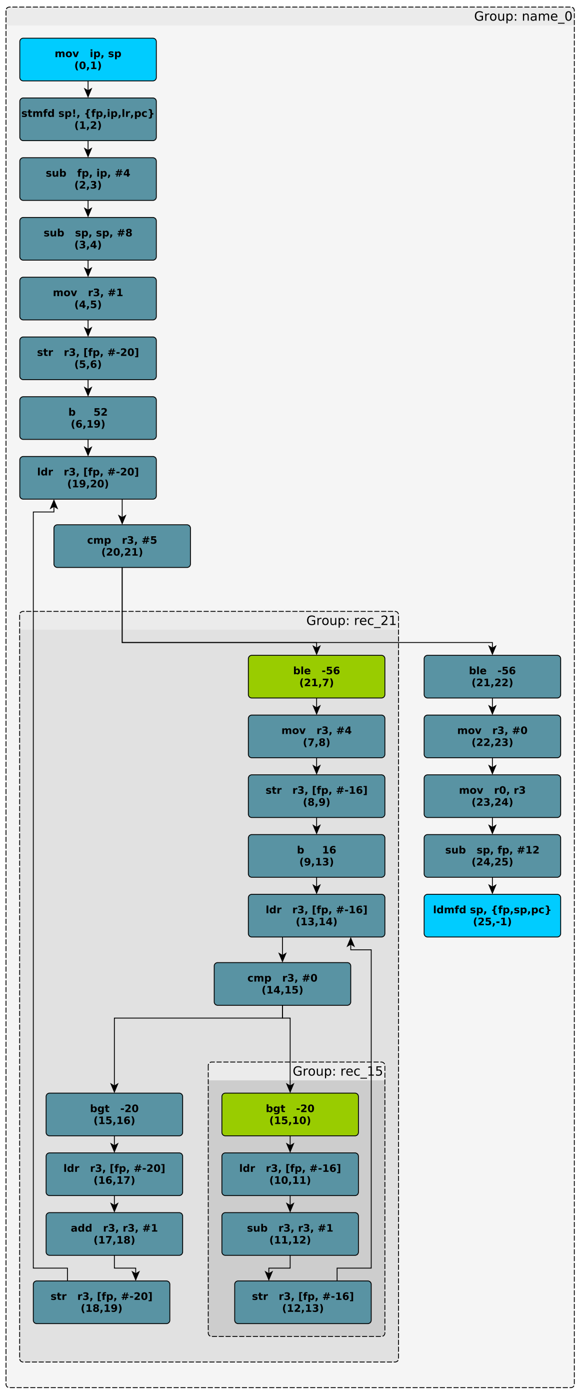

Figure 1: Example of a dependency graph with nested loops