

MCVT - User Manual

Raya Leviathan, Ira Gordin

August 22, 2004

`raya.leviathan@weizmann.ac.il, ira.gordin@weizmann.ac.il`

Abstract

This manual describes how to install and use the MCVT tool, to validate the translation of C programs, produced by the SCADE code generator, and translated by WindRiver DiabData C compiler, for the power-pc processor.

1 General Description

MCVT tool reads a synchronous C program and its translation into machine code. The C program is one synchronous C function, the *source program*, and the machine program is the disassembly of one function extracted from a binary executable file, the *target program*. The binary file is the executable produced from the C program by the DiabData compiler.

MCVT extracts the formal semantics of the two functions, and checks that the translation is correct. The correctness is checked for each of the observable variables of the function at a time, and one message, `Valid` or `Invalid` is produced for each observable variable. See [LP04].

2 Installation and Invocation

2.1 Installation

unzip the file `MCVT_DIR.zip`. A directory named `<MCVT_DIR>` is produced. Type:

```
set path = ( <MCVT_DIR full path>/Solaris $path)
```

or

```
set path = ( <MCVT_DIR full path>/Linux $path)
```

according to the host machine.

This will set the `$path` environment variable to include the full path name of `<MCVT_DIR>` directory. The installation package contains the following directories:

- MCVT_DIR
- MCVT_DIR/bin - Contains all binaries and executable files. The `$path` environment variable should be set to contains this directory.
- MCVT_DIR/license - Contains the LICENSE file of CVC.
- MCVT_DIR/example - Contains an example. To verify the installation, type `run_tv.pl com_3.c com_3 com_3` in this directory.

2.2 invocation

Step 1 - Running C preprocessor: MCVT does not recognize the C preprocessor directives. Thus the preprocessor should be first invoked. Let `<file.c>` be the C input file. first run the preprocessor:

```
dcc -E file.c -o file1.c
```

Step2 - Compilation: In order to validate the translation, a binary file should be produced. The compiler is invoked with the `"-g1"` flag, to produces symbolic information. The `"-O"` flag can be used as well. The `"-g1"` flag ensure that if instructed, the compiler produces optimized code. The binary file that is provided to MCVT is an executable file, and not an object file (i.e the output of the loader - `ld`). An empty `main.c` file can be provided, if needed. for example, the compiler may be called with the following arguments:

```
dcc file1.c main.c -g1 -O -o file1
```

Step3 - Creating disassembly: To produce the disassembly file, DiabData disassembly utility is invoked as follows:

```
rtasim -il <function-name> <executable-file> > <node-name>.dis
```

Step4 - Running MCVT:

```
run_tv.pl <source-file> <executable-file> <node-name>
```

Where:

<source-file> is C source file that contains C code of the function to be checked.

<executable-file> The executable file name. The executable should contains the binary code of the node.

<node-name> The name of the function to be checked. The disassembly file name should be `<node-name>.dis`

2.3 output files

The directory `sctvtmp` contains all CVC input files. One file for each observable variable. The file `run_tv.log` contains the output of the run. All other produced files are for MCVT debug purposes.

2.4 Required Software

- `rtasim` - DiabData disassembly
- `perl` - The perl (Practical Extraction and Report Language) interpreter
- `cvc` - The Coperative Validity Checker provided with MCVT, (see the enclosed license).

3 Synchronous C syntax

We define here the syntax of one function, since MCVT validate the translation of one function at a time.

- A function should have the following form:

```
node ::= void <function-name> (< type-name> *_C_) {  
    <declarations>  
    <statements>  
}
```

- The only allowed basic types are: `int`, `float`
- The following statements are not allowed: `case`, `while`, `do`, `for` `goto`, `continue`, `break`, `return`.
- No assignment to pointer is allowed. For example, the following is not allowed:

```
tstruct1 *s1;  
struct1 *s2;  
s1=s2;
```

- The address operator `&` is allowed only when applied to actual parameter of a procedure. In this case, the produced transition is assigning an uninterpreted function value to all the variable which its address is taken, or as multiple assignment to all the fields of the structure whose address is taken.
- Function calls - currently ignored (no message is printed).
- No support for floating point constants.

If a limitation is violated, an error message is printed, and validation is stopped.

4 The machine instructions

The following are the implemented machine instruction. Some of them are only partially implemented. When an unimplemented instruction is encountered, an error message is announced. The supported machine instructions:

- fadds
- addi
- addis
- add
- andc
- mullw
- mulli
- fmuls
- stfs
- stwu
- stw
- stmw
- bclr
- mfspr
- lfs
- lwz
- lwzu
- lmw
- cmpli
- fcmpl
- cmpi
- b
- bc
- fsubs
- subf
- cntlzw
- rlwinm_
- rlwinm
- mfer
- neg
- orx
- divw

5 Validation process

The validation process is composed of the following stages:

1. `tvppc` produces state transformer system from the disassembly (target system), which is written to the file `a_sdtout0`

2. `tvc` produces state transformer system from the C source (source system), which is written to the file `c_stdout0`.
3. `seperate_files.pl` Decomposes the verification conditions file. For each observable variable, a separate verification file (`*.cvcin`) is produced, in the subdirectory `sctvtmp`.
4. `cvc` checks each verification file, separately. These runs are the most time consuming stage.

If `cvc` run was completed with `Valid` for all the verification files, the translation is valid.

6 Handling invalid translation message

If the validation was failed for one of the variables, there is no indication concerning the reason of the fail. Please send the example to one of the authors.

References

- [LP04] I. Gordin R. Leviathan and A. Pnueli. Validating the translation of optimizing industrial compiler. In *Proc. of 2nd Internation symposium on Automated Technology for Verification and Analysis, LNCS (to be appeared)*. Springer-Verlag, 2004.