DEVELOPMENT OF VERIFICATION TESTING CAPABILITIES FOR SAFETY CODES

D. L. Aumiller, G. W. Swartele, J. W. Lane, F. X. Buschman and M. J. Meholic
Bettis Atomic Power Laboratory, West Mifflin, Pennsylvania, United States

david.aumiller.contractor@unnpp.gov

ABSTRACT

The analysis codes that are used to assess the safety posture for light water reactors can be quite complex in terms of the underlying physics. Additionally, the computer codes tend to include many features resulting in programs that are often in excess of 105 lines of source code. Thus, there is a need for automated verification techniques for such codes.

This paper will describe the techniques that have been developed for verification testing of an in-house version of the COBRA-TF (hereafter referred to as COBRA) analysis code. These techniques rely on the ability to adequately test for null changes in code results for a large problem suite. This paper will present information on the techniques used to perform the null test and the automated null testing process. Descriptions of how this testing capability has been used to test the restart capability and timestep backup logic in COBRA are provided. Finally, the use of quantitative verification techniques is presented.

1. INTRODUCTION

The development of reactor safety codes such as TRACE, RELAP5-3D and COBRA (References 1, 2 and 3) require the development of verification techniques to ensure that the numerical techniques, models and correlations have been properly implemented. The verification processes used in the formal deployment of these codes can represent a significant portion of the life-cycle costs associated with the development and maintenance of these codes.

Often the costs associated with the continued verification of the code are attributed to the lack of an engineered system that can be used to perform the required quality assurance tasks. Instead of providing a methodology to verify safety codes using a comprehensive system with quantitative metrics, often times the verification is based on a staged approach. In this approach, new features are verified when installed and the results from one version of a code are compared to the previous version of the code for a number of problems and any differences in the results are reconciled.

An automated verification system has been developed for use with the COBRA analysis code. The characteristics of this verification system have been engineered to provide verification testing that goes beyond the techniques historically used for reactor safety analysis codes. The characteristics of the system are:

1) Coverage matrix providing evidence of which problems test which features
2) Automated methods to perform all of the required testing
3) Use of quantitative installation problems

4) Development of a set of solution metrics that can detect small changes in the solutions
5) Capability to perform automated null testing
6) Ability to test the adequacy of the restart process
7) Ability to test the adequacy of the timestep backup process

The following sections of this paper will describe these characteristics in more depth and describe how they have been implemented in the context of the COBRA code.

2. COVERAGE MATRIX

At the heart of the automated verification process is a suite of test problems that has been developed to exercise all of the supported features in COBRA. As a means to ensure that the features are properly tested, a coverage matrix has been developed. A sample of the verification coverage matrix is shown in Table 1.

Table 1- Sample Portion of Coverage Matrix
(Note – Some columns have been omitted for space)

				Test Problems																															
	Option Tested	Tested	Restarted	2d_gap	beta_prob	boil_onset	ccfl & ccfl_mdot	chf	conduction	connect_test	cpl	cpl_mom	cmt_gap	dam_break	delta_p	delta_p_spv	faucet	fill	flecht	hstrat	init_fl_cond	inj	jet_injection	large_pipe	leak_path	namgap_gas	par_fill & par_fill2	ss_init	sstemp	symmetry	tmin	tube	unwet	utube	waterfall
Tmin	Model 0	Yes	Yes			R			X		R								X									X	X		X	X	R		
	Model 1	Yes	Yes		R																										X				
	Model 2	Yes	No																												X				
	Model 3	Yes	No																												X				
	Model 4	Yes	No																												X				
	Model 5	Yes	No																												X				
	Model 6	Yes	No																												X				
	User Specified	Yes	No																												X				
Boundary Conditions	Type 1	Yes	Yes	R	R	R	X	X	X	X					X		X		X		X	R		X		R		X	X	X	X	X	R		
	Type 2	Yes	Yes		R	R		X	X	X	R						X		X		X			X		R		X	X	X	X	X	R		
	Type 3	Yes	Yes				X						X							X		R	R			R									X
	Type 4	Yes	Yes	R			X						X											X											
	Type 5	Yes	Yes	R									X										R		R										
	Type 6	Yes	Yes																			R													
	CCFL	Yes	Yes				R																												
	Variable Axial Momentum Area	Yes	Yes								R																								
	Dual Mass Source	Yes	Yes	R																															
	Axial Leakage Path	Yes	Yes																						R										
	Gap Leakage Path	Yes	Yes																						R										
	Zero cross-flow	Yes	Yes										X																						
	Use of Functions	Yes	Yes	R											X																X				
	Use of 0 or 1 Void in BC	Yes	Yes	R	R		X	X	X				X		X				X		X	R						X		X	X	X	R		
	Read values from file	Yes	Yes																			R													
	Coupled	Yes	Yes								R	X																							
	Coupled - default ae	Yes	Yes								R																								
	Coupled - no ae	Yes	Yes								R																								
	Coupled - specified drop size	Yes	Yes								R																								
	Control System Coupling	Yes	Yes								R																								
	Coupled - specify non-zero ae	Yes	Yes								R																								
	Kinetics Coupling	Yes	Yes								R																								
	Jet Injection	Yes	Yes																				R												

The coverage matrix serves two purposes. The first is to definitively show that the all of the code features or input options are exercised in the verification suite. The second purpose of the matrix is to provide a clear indication of which features are tested in each problem. As will be described later, the verification testing procedure has the ability to look for programming errors associated with repeating a timestep, restarting the solution from an intermediate time point or when performing code modifications that should not impact results. When a difference is found as part of testing, the code matrix can be used to identify the commonality between test cases to more quickly isolate problems.

As new code features are added to the code, the coverage matrix is modified to include them. The "Tested" and "Restarted" columns in the table are calculated such the addition of the new feature as a row would indicate a "No" until a new verification problem is provided. Presently, the matrix is manually maintained; however, a process to build the matrix automatically from the input decks is currently being developed.

3. AUTOMATED METHODS

The execution of the automated verification suite for COBRA requires 262 executions of the COBRA analysis code. Of these executions, 124 problems are used to test input failures. With such a large number of executions the ability to perform parallel executions of the code can significantly reduce the overall execution time. The make utility has been chosen as the driver for the automated process. This choice was predicated on the following:

- It can perform parallel operations and requires no setup of this feature
- Its use of targets and dependencies can be exploited to identify sub-sets of the problems that can be built up to form the entirety of the required testing
- The dependencies naturally allow for staging of executions. For instance, before a case can be restarted, the base case must be executed.
- It is widely available on computing platforms
- It provides an extensible platform such that new cases are easily added.

The importance of the last point should not be overlooked. The process should be easy to modify to simplify maintainence and to assist in the addition of new problems In our system, problems can be added to the automated suite by simply adding two files (one for input and one for post-processing) and then making a one-line change to the master Makefile.

To be most useful, an automated verification process must be capable of running in a reasonable time period. Using 12 processors, the COBRA verification process is capable of being executed in under 10 minutes. As a result, the verification suite can be executed many times during the development of new features allowing potential issues to be identified early in the development cycle and thereby saving development resources.

Another important aspect of the automated verification process is that it has been developed to clearly identify any code failures that may occur during an execution. If the process completes all of the requested simulations properly, the string "NO FAILURES DETECTED FOR verification" is displayed. Otherwise, a list of the cases that failed is displayed.

4. QUANTITATIVE INSTALLATION PROBLEMS

All reactor safety and analysis codes have a suite of problems which are executed as part of the normal code development and deployment process. Often these problems are used in one of two ways. The simplest use of these problems is as a "go no-go" gauge; meaning that the only thing that is really important is whether the code reaches the end of the simulation. Most frequently, there are qualitative assessments performed on the simulations, with the most common being looking at the ASCII output file to look for differences with a previous version.

To provide better verification, quantitative assessments are used in the COBRA process. This is performed by comparing code predictions to independent calculations of the parameters of interest, such as heat transfer coefficients, critical heat flux (CHF), minimum film boiling temperature (T_{min}). An example of such a quantitative comparison can be found in Figure 1 which compares the results of all of the correlations for T_{min} as a function pressure with independent calculations. Figure 1 is one of almost 100 figures that are generated from 88 different problems as part of the COBRA automated verification process and which are then reported with every code release.

In addition to verification of correlations, a number of canonical problems are included in the automated verification suite to ensure that when available, the proper analytical solutions can be reproduced. Examples of such canonical problems include filling a stack of volumes, the "faucet" problem of water falling through air and the solution of the transient conduction equation with constant boundary conditions. Figure 2 shows such a figure which shows that the steady-state solution to the conduction equation is obtained for solid cylinders, slabs and for annuli with convection at both the inside an outside surfaces.

The use of quantitative metrics has been valuable in the standard code development process. As new code features are added to COBRA, the validity of the previous portions of the code can be quickly established. As new capabilities are added to the code, it is a requirement that quantitative verification problems be added to the verification suite.

5. SOLUTION METRICS

As will be discussed in the next section, it is often desired to be able to know if two solutions are the same. To accomplish this, a method of collapsing the solution variables to a few, easily compared metrics is required. This has been accomplished in the COBRA system through the verification data files. The verification data files, which can be written at arbitrary solution times, contain a header with the following information:

- Code version number
- Compilation date and time
- Job Execution time

For each time the data is requested the current timestep size is printed along with the value of selected variables summed over the entire domain. The variables are printed to machine precision in both scientific notation and with the hexadecimal representation. A sample verification data file has been included in Figure 3. The variables used in COBRA are:

- pressure
- steam partial pressure
- three phasic axial mass flow rates
- three phasic transverse mass flow rates
- powered heat structure temperatures
- non-powered heat structure temperatures

These parameters have been chosen to indicate if changes have been made to either hydraulic or heat transfer portions of the code. By providing the sums of the variables instead of the $\mathcal{L}_1$ or $\mathcal{L}_2$ norm provides the ability to detect the situation where the magnitude of the flow rates was unchanged but the sign changed. The $\mathcal{L}_\infty$ norm was not selected since it ignores the entire solution domain except the location where the parameters are at their maximum.

6. NULL TESTING CAPABILITY

During the life cycle of a reactor safety analysis tool there are many times when it is desired to be able to show that the solutions from two code executions are identical Examples of such events include comparing the results using the same executable on different computers, when adding new features to the code and when performing code restructuring. To provide a means of ensuring the same result between different versions of the code a null testing capability has been included in the verification suite.

A null test comparison is performed by the use of a specific target in the Makefile. The target runs all of the verification test problems. All of these test problems include the writing of a verification data file. Often times it is desired to compare several versions of the code to a fixed standard. As such, the ability to develop a reference set of solutions exists. When the target for the reference solutions is selected, all of the verification data files from the verification problems are copied to a known location. To perform a null test, the verification suite is re-executed with a different version of the code. The verification data files from this execution are then copied to a common directory. At this time, the solution data from the reference and new files can be compared. If no solution differences are detected, the unambiguous message "NO DIFFERENCES DETECTED IN BINARY COMPARISON FOR VERIFICATION". If differences are detected in the string "BINARY COMPARISON FAILED FOR VERIFICATION" is displayed with a list of the problems which included a difference. The list of problems can then be used in conjunction with the Coverage Matrix to identify the likely cause of the unexpected differences.

7. RESTART TESTING

Thermal-Hydraulic analysis codes such as COBRA contain the ability to save the solution variables at a given point in an analysis such that the calculations can be restarted. In order to have confidence in the restart capability, the restart process should be demonstrated to not impact the solution. This capability has been added to the COBRA verification process and is performed by default. The authors are unaware of any other comprehensive and quantitative restarting testing capability utilized by other reactor safety analysis codes.

This comparison is performed by running some of the calculations (designated with an "R" in the coverage matrix) twice. In the first calculation, the problem is run in its entirety. The calculation includes the creation of a restart file at some intermediate time. A second solution is then performed where the restart file that was written is read and the solution is continued to the same end time. Once both solutions are complete, the verification data files are compared. The use of the verification data files ensures that, to the extent possible, we compare the results to the precision of the computations and do not simply rely on the standard output which is typically printed to 4-5 significant figures.

In a manner similar to the null testing, described above, when restart testing is performed the developer receives either a message stating that no differences were encountered in the restart comparisons or they receive a failure message along with the list of problems which had differences.

Since all features should have the ability to be successfully restarted, it is important to track the restart testing in the coverage matrix. The verification problems have been developed such that all of the features can be tested in 21 different problems. It is important to note that it is acceptable for some items to not be restarted. For these situations the verification documentation should clearly identify the reason for the lack of testing. An example of this, as shown in Table 1, not all of the T_{min} correlations are restart tested. In this example, there is a single variable that contains the information concerning which of the correlations has been selected. Since two of the models have been restarted, the variable which stores the correlation must be properly stored and used in the restart process.

8. BACKUP TESTING

The solution process for thermal-hydraulic analysis codes can result in a solution for a given timestep that does not meet the metrics required to accept the timestep size solution and carry and proceed to the next timestep. These metrics can be related to the iterative convergence behaviour, mass and energy error estimates or on detecting large rates of change in parameters such as pressure and void fraction. If a solution is deemed to be unacceptable, codes such as COBRA will typically return the solution state to the end of the previous timestep size and either change the solution algorithm or proceed with a reduced timestep size.

The process of restoring the values to their values at the previous timestep value should be demonstrated as part of the normal code verification process. Care must be taken to save parameters other than the simple independent solution parameters in order to completely replicate the solution state. Items that also need to be addressed include parameters that are time-smoothed (e.g. drag and heat transfer coefficients), parameters involved with specialized models (moving quench front and choked flow) and solution variables used in setting the timestep size. The authors are unaware of similar quantitative and comprehensive timestep backup testing that is performed as part of standard reactor safety analysis code development.

The ability to perform timestep backup testing has been developed for use with COBRA. This testing is performed with two different modes of backup related to the two methods in which timestep sizes can be forced to backup; one in an inner loop and one in a timestep loop. In this testing, a problem is run three different times. The first execution is a standard execution. In the second and third execution, the code is artificially forced to backup with both methods at an input specified timestep number and repeated for an input specified number of attempts. At the end of the three executions, the verification data files from the two executions with the artificial backups are compared to the standard execution. In a similar fashion to the restart testing, when the backup testing is performed the developer receives either a message stating that no differences were encountered in the backup comparisons or they receive a failure message along with the list of problems which had differences.

9. CONCLUSIONS

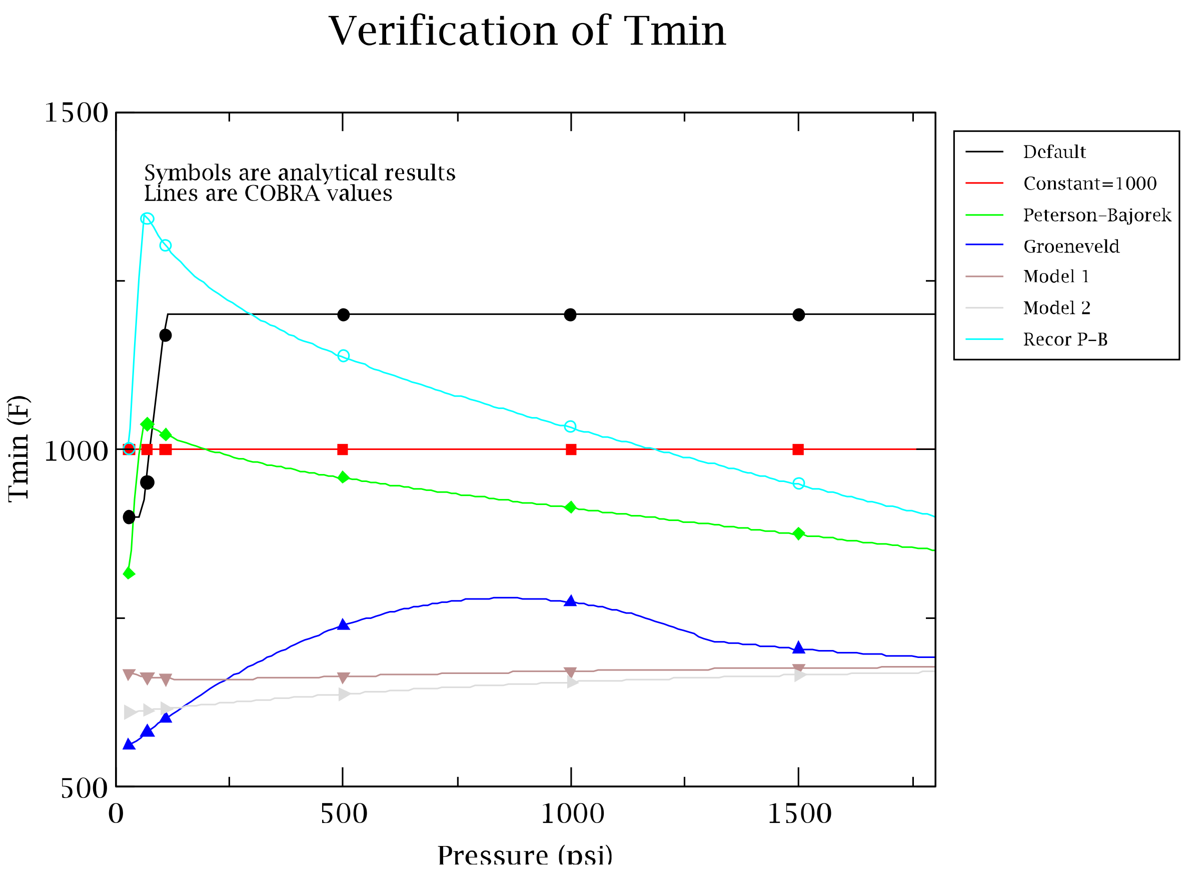
This paper provides a description of various verification testing methods that can be applied to reactor safety analysis codes such as COBRA, TRACE and RELAP5-3D. The paper describes the unique, comprehensive and quantitative testing features that have been implemented in the COBRA development process. The testing needs that have been addressed in the context of COBRA are equally important for other codes and the same techniques could be applied to those codes to help demonstrate the proper implementation of the restart and backup capabilities. The creation of tools such as those described in this paper form an important part of the Quality Assurance processes for reactor safety analysis tools. They require resources to design, develop and implement; however, when properly implemented they significantly enhance the code development and maintenance process and help establish the required confidence in these complex analysis tools.

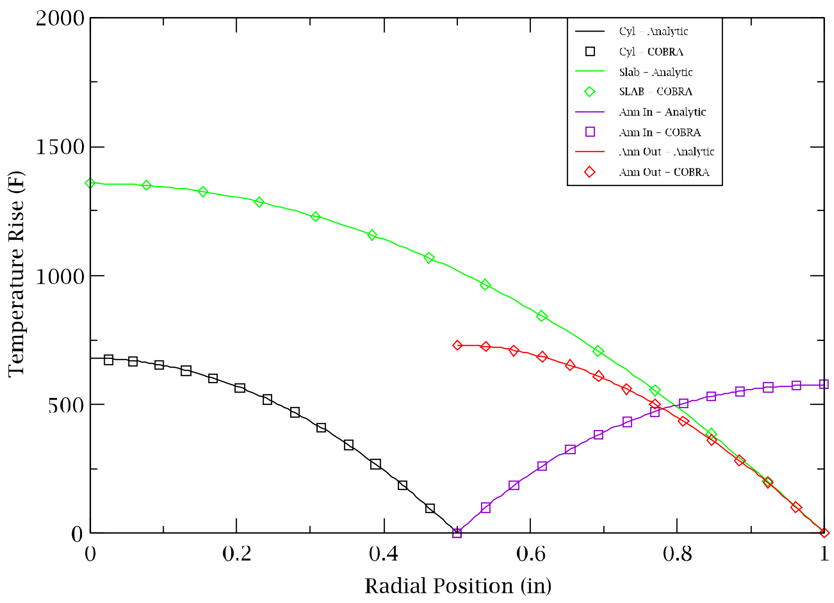
10. REFERENCES

[1] "TRACE V5.0 Theory Manual", TRACE V5.0p1, United States Nuclear Regulatory Commission (2008)

[2] "RELAP5-3D Code Manual," INEEL-EXT-98-00834 (Volumes 1-5), Rev. 2.6, Idaho National Laboratory (2007).

[3] Thurgood, M., et al., "COBRA/TRAC: A Thermal Hydraulic Code for Transient Analysis of Nuclear Reactor Vessels and Primary Coolant Systems," NUREG/CR 3046 (Volumes 1 5), Pacific Northwest Laboratory (1982)


Verification of Tmin
Symbols are analytical results
Lines are COBRA values
Tmin (F)
Pressure (psi)
1500
1000
500
0
500
1000
1500
Default
Constant=1000
Peterson–Bajorek
Groeneveld
Model 1
Model 2
Recor P–B


Temperature Rise (F)
Radial Position (in)
2000
1500
1000
500
0
0
0.2
0.4
0.6
0.8
1
Cyl - Analytic
Cyl - COBRA
Slab - Analytic
SLAB - COBRA
Ann In - Analytic
Ann In - COBRA
Ann Out - Analytic
Ann Out - COBRA

```
Verification Data File

 COBRA Program Info:
 Compile/Load Date-Time -   20121023  10:13:59
 Version - 2.2.0

 Job Info:
 Date  - 10:23:2012
 Time  - 10:45:58
 Title -                                                    u-
tube
 Host  - b11lc490



 Verification Data Edit

 transient time =   8.000000000E+00   step number =
832

 step size =   9.0954032356496128E-03
3F82A09C5B874400

 Verification summation data:

  p     active sum =   8.9447777131446583E+02
408BF3D279C454C8
  p_st  active sum =   1.2416133447376694E+02
405F0A534DD422F8
  p_ncg active sum =   7.7031666541094751E+02
4088128887DFFE5A
  fem   active sum =   4.4975551555785616E-06
3ED2DD368C2AF7AB
  fgm   active sum =   6.7848895413841186E-03
3F7BCA78EAFED77E
  flm   active sum =   2.8568796039111950E+00
4006DAE3B19B439E
  wem   active sum =   7.2433804192735666E-08
```

Figure 3 - Example Verification Data File