INL/CON-14-31184
PREPRINT

AUTOMATED, HIGHLY ACCURATE VERIFICATION OF RELAP5-3D

Proceedings of the 22nd International Conference on Nuclear Engineering ICONE22
July 7-11, 2014, Prague, Czech Republic

George L. Mesina
David L. Aumiller
Francis X. Buschman

July 2014

The INL is a
U.S. Department of Energy
National Laboratory
operated by
Battelle Energy Alliance

Proceedings of the 22nd International Conference on Nuclear Engineering
ICONE22
July 7-11, 2014, Prague, Czech Republic

ICONE22-31153

AUTOMATED, HIGHLY ACCURATE VERIFICATION OF RELAP5-3D

George L. Mesina
Idaho National Laboratory
Idaho Falls, ID, USA

David L. Aumiller
Bettis Laboratory
Pittsburgh, PA, USA

Francis X. Buschman
Bettis Laboratory
Pittsburgh, PA, USA

ABSTRACT

Computer programs that analyze light water reactor safety solve complex systems of governing, closure and special process equations to model the underlying physics. In addition, these programs incorporate many other features and are quite large. RELAP5-3D[1] has over 300,000 lines of coding for physics, input, output, data management, user-interaction, and post-processing. For software quality assurance, the code must be verified and validated before being released to users. Verification ensures that a program is built right by checking that it meets its design specifications. Recently, there has been an increased importance on the development of automated verification processes that compare coding against its documented algorithms and equations and compares its calculations against analytical solutions and the method of manufactured solutions[2]. For the first time, the ability exists to ensure that the data transfer operations associated with timestep advancement/repeating and writing/reading a solution to a file have no unintended consequences. To ensure that the code performs as intended over its extensive list of applications, an automated and highly accurate verification method has been modified and applied to RELAP5-3D. Furthermore, mathematical analysis of the adequacy of the checks used in the comparisons is provided.

INTRODUCTION

The state-of-the-art nuclear reactor system safety analysis computer program developed at the Idaho National Laboratory (INL), RELAP5-3D, continues to adapt to changes in computer hardware and software and to develop to meet the ever-expanding needs of the nuclear industry. To continue at the forefront, code testing (Verification and Validation) must evolve with both code and industry developments.

A form of validation testing, Developmental Assessment[3] (DA), is applied to validate RELAP5-3D performance for generic nuclear safety analyses. DA checks RELAP5-3D calculations against analytical solutions, separate-effects and integral tests, and whole plant data. First performed in 1990, each new RELAP5-3D code release is reevaluated with DA. Verification checks coding against its documented algorithms and equations and compares its calculations against analytical solutions and the method of manufactured solutions. While traditional verification performs these checks and comparisons for every code release, sequential verification performs checking only for the original code release but continually compares calculations between consecutive versions on the test cases. If unexplained differences are not detected, the new release is sequentially verified. During its life-cycle, RELAP5-3D has employed sequential verification, expanding its test set with each new feature and capability. As the test set grew and greater fidelity was sought, the 'diffem' utility was developed that, for every input file in the test set, performed a character-by-character comparison of the output files of two versions of RELAP5-3D. Thus every difference recorded on the printed output files was automatically detected.

This method has deficiencies. Differences in calculations between two versions may go undetected because insufficient decimal digits are recorded on the output file. Additionally, code changes that are intended to modify results can cause unintended changes to physical models that are missed in the verification process. Analysis shows that some code features and capabilities, including restart and backup, were not checked by existing test suites.

To address these deficiencies, state-of-the-art verification techniques[2] have been adapted and extended to the RELAP5-3D code. The method provides sufficient coverage by creating a comprehensive verification test suite. A new verification file, that compresses key data onto small disk files to improve detection, has been developed for RELAP5-3D. The method's *null testing* compares the verification files of two different code versions for the same test case. It extends the original scope of verification when applied to a single version to test its restart and backup processes.

The results of implementing and analyzing this method are presented. Section 2 explains the detection of differences and the new verification file. Section 3 details the coverage of code

features. Section 4 defines the Tests/Features Matrix that visualizes coverage. Section 5 discusses null testing. Section 6 explains restart testing and summarizes the resulting code improvements. Section 7 discusses backup testing. Section 8 explains the method's programming and automation. Section 9 statistically analyzes this form of testing.

NOMENCLATURE

H_0	Null Hypothesis
H_1	Alternative Hypothesis
N	Number of test cases
P	Probability function
S	Size of the verification file
T	Temperature
T_r	Trip
UP	User-reported Problem
u_f	Liquid internal energy
u_g	Gas internal energy
V_f	Velocity of the Liquid (Fluid)
V_g	Velocity of the Gas
X	Random Variable for the Hypothesis Test that two runs are identical
X_a	Noncondensable Quality
X_i	Random Variable for Test Case i, the i^{th} input deck.
Y	Control Variable
α	significance level of a hypothesis test
α_g	Void fraction of gas
β	Probability of committing Type II error t
Δp	Pressure Drop
Δt	Timestep for Hydrodynamic Advancement
Δt_{kin}	Timestep for Neutron Kinetics Advancement
ε	Sum of RELAP5-3d estimate errors
φ	Neutron Flux
$\boldsymbol{\rho}_b$	Density of Boron

DIFFERENCE DETECTION AND VERIFICATION FILE

Detection of any difference between runs of an identical input file by two different versions of RELAP5-3D can be guaranteed by writing all data in RELAP5-3D memory on a binary file and comparing them. This is impractical and costly:

A. Since modeling detail and number of writes control the size of the output file, it can grow without bound.
B. Such writes can greatly increase code runtime.
C. To guarantee detection, every new variable added to the code must be added to the write statements.

Even with powerful compression techniques, a user could inadvertently overfill disk space. Two principle ways to reduce size are to restrict the frequency of writes and reduce the data written. Towards this end, consider that there are three major categories of data/variables:

1. *Primary variables from the five physical phenomena: heat transfer, thermal hydraulics, neutronics, controls and trips. See Table 1.*
2. *Secondary variables derived from primary variables and used in constructing the set of equations solved for the primary variables on the next advancement. E.G. heat capacitance, enthalpy, power.*
3. *Output-only variables. Do not feed back into primary or secondary variables. E.G. water packer count.*

Table 1. RELAP5-3D Primary Variables

Quantity	In manual	On file
Pressure	p	P
Liquid internal energy	u_f	Uf
Gas internal energy	u_g	Ug
Void fraction of gas	α_g	VOIDg
Noncondensable quality	X_a	QUALa
Density of boron	$\boldsymbol{\rho}_b$	Boron
Liquid velocity	V_f	Vf
Gas velocity	V_g	Vg
Heat Structure Temperature	T	Temp
Neutron flux	φ	Flux
Timesteps sum	Δt, Δt_{kin}	dtsum
Trips	T_r	Trips
Control system value	Y	Cntrl

If a primary variable is different between two calculations, variables in category 2 and 3 that are derived from it will also differ. If a secondary variable differs, it will affect the equations of (at least) one primary variable and thus its value when the equation is solved. Differences in output-only variables do not affect category 1 or 2 variables. Writing every category 1 and 3 variable on the file would guarantee that every difference is detected, but issues A and C remain.

Writing only Category 1 variables eliminates issue C, but allows a few differences to escape detection. To overcome issue A, the L_1-norm of each primary variable is written on the verification file. The TH system solution and RHS are included to aid debugging, and sums of appropriate output-only variables, namely error measures, reduction counts, and backup counts, are included to further reduce the possibility of missing differences. Sums are calculated in quadruple precision to preserve the first 32 decimal places[4] of the L_1-norm and are written in both scientific notation and z32 hexadecimal so that all bits of the sum are represented.

The information written to the verification file on a given advancement is referred to as a verification kernel. Kernels are numbered and record the advancement count and cumulative time. Input cases are numbered and their titles are recorded. Each kernel is smaller than 1.4 KB. Writes to the file are terminated when it reaches 1 MB. Activation, frequency and

duration of kernel output are controlled by input. In addition to its use in the automated verification process, the kernel information can facilitate debugging. A sample verification file with 1 input case and 2 kernels is shown in Fig. 1.

The small size allows storage of many such files for future comparison; therefore unique internal identification data is necessary, including: code version and compile time, date and time of run, CPU time, and name of the computer on which the case was executed. This ID data should be ignored when comparisons are made.

Figure 1 Example of a Verification File

```
RELAP5-3D/Ver:4.1.3   inl425834-linux
Time compiled: Jan 14 2014 13:00:36
Date and Time of run: 14/01/14     13:03:26

Case  1  edhtrkm: edward's pipe problem base case with extras

Kernel      1    Advancement=      110 time=  1.0100E-01
P=        4.936446436625647OE+07 4018789EF82EE17DF400000000000000
Uf=       1.964548745003424 8E+07 40172BC42F733571C840000000000000
Ug=       5.452133793790415 9E+07 40189FF6FCF80D3E5500000000000000
VOIDg=    7.068407737354172 9E-00 4001C460CAD8AEB11000000000000000
QUALa=    0.0000000000000000E+00                                0
Boron=    0.0000000000000000E+00                                0
Vf=       2.0609367404278057E+02 40069C2FF60B4C740DFA000000000000
Vg=       2.3448536361144574E+02 4006D4F881944BA7E45F000000000000
RHSth=    7.5475342950407861E+04 400F26D357CB9911E9C0000000000000
SOLth=    6.7091414384136505E+04 400F06136A15142A4524000000000000
Error=    9.2690379142344780E-05 3FF184C58969992D0000000000000000
Temp=     1.1047787599936748E+05 400FAF8DE0417E9AA9B3E00000000000
Flux=     6.9442838715461731E+10 402302B1DA0BB7634000000000000000
dtsum=    3.0000000000000001E-03 3FF689374BC6A7EFA000000000000000
Trips=    3.1979851464308495E+00 4000995793C881D39700000000000000
Cntrl=    3.7058322760846792E+06 4014C45F42356BE26176967E4A6D999A
Rdc:Crnt,Extrp,Mass,Prop,Qual=      0      2      0      2      0
Rpt:Air,DelP,Flip,Jpack,Vpack=      0      0      0      0      0

Kernel      2    Advancement=      509 time=  5.0000E-01
P=        1.1610017826645510E+07 4016624F43A73E14910000000000000
Uf=       1.3706563288738387E+07 4016A24A8693D5848F80000000000000
Ug=       5.3792556235066078E+07 40189A67961E16A52E00000000000000
VOIDg=    2.0127747744320381E+01 4003420B4137FFE69F80000000000000
QUALa=    0.0000000000000000E+00                                0
Boron=    0.0000000000000000E+00                                0
Vf=       2.8891214895245306E+02 400720E98297FFDDD508000000000000
Vg=       9.1675057057330686E+02 4008CA6012B250D8B1C0000000000000
RHSth=    7.5475342950407861E+04 400F26D357CB9911E9C0000000000000
SOLth=    6.7091414384136505E+04 400F06136A15142A4524000000000000
Error=    1.0335807997476587E-04 3FF1B183E4B07057900000000000000
Temp=     1.0939814425864998E+05 400FAB5624EE2287CFAFD00000000000
Flux=     2.7820142401306078E+07 4017A8806E66BBFEC000000000000000
dtsum=    3.0000000000000001E-03 3FF689374BC6A7EFA000000000000000
Trips=    2.3019990000000000E+00 400026A7E73A365CB400000000000000
Cntrl=    8.6401335729741725E+05 4012A5E1AB6EFAFE4088D0CAE5400000
Rdc:Crnt,Extrp,Mass,Prop,Qual=      0      2      0      2      0
Rpt:Air,DelP,Flip,Jpack,Vpack=      0      0      0      0      0

CPU Time=  2.4700000000000000E-01 size    2773
```

CODE FEATURE COVERAGE

Rather than traditional coverage, which determines the percentage of lines of code exercised by a test suite, each code feature used in the modeling of nuclear power plants must be tested for sequential verification. Since so many features exist, a judgment was made to initially exclude those not used at all in the DA and other test suites or rarely used for nuclear power models. Therefore, coverage is not 100% of all code features, but important capabilities are verified.

Important categories of code features and models include: hydrodynamic components, volume and junction options, heat structure types, correlations, boundary conditions, trips, tables, control variables, reactor kinetics, fluids and Appendix K. User choices affect the way the code operates and are also included: certain card-1 options, time advancement schemes, solvers, and several others. Among excluded features are: most fluids and card-1 options, debug and special output control, and the mechanistic GE separator component.

FEATURES TESTS MATRIX

The verification test suite is comprised of 43 input decks that test the selected features. Most of these have multiple cases which vary one feature, e.g. heat transfer mode. If a feature is tested in one kind of component, it may not be tested in another. For example, if CCFL is tested in a pipe, it need not be tested in valves. Input decks include problems with analytical or well-known solutions, such as, control variable functions that have exact mathematical valuations.

The actual Features Tests Matrix[2], with over two hundred rows and 46 columns, is broken into 6 sub-tables; one is shown in Figure 2.

Figure 2. Sub-table 1 of the Features Tests Matrix

Features	Present	Restart	2phspump.i	3dflow.i	ans.i	boronm.i	crit.i	cyl3.i	duklerm.i	eccmix.i	edhtrkm.i	eflag.i	enclss.i	fric.i	fwhtr.i	gota27.i	hse.i	httable.i	httest.i	hxco2m.i	jetjun.i	jetpmpm.i	12-5-emA.i	131acc.i	neptunus20m.i	pack.i	pitch.i	radialm.i	rcpr.i	refbunm.i	reflecht.i	regime.i	rigidbodym.i	rthetam.i	rtsampnm.i	rtsamppm.i	slab3.i	sphere3.i	state.i	todcnd.i	turbine9.i	typ12002.i	typ_kindt.i	valve.i	varvol2.i
	#	#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
Hydro-Component																																													
SNGLVOL	X	X	X						X			X			X						X	X	X						X						X	X					X	X	X		X
TMDPVOL	X	X	X	X		X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	X	X	X
SNGLJUN	X	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X	X	X			X	X	X	X			X	X	X	X	X
TMDPJUN	X	X	X			X		X	X	X			X	X	X	X		X	X	X	X	X	X		X			X			X	X	X	X	X	X	X	X				X	X		
PIPE	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X		X	X			X	X	X			X	X	X	X	X	X	X	X	X	X	
ANNULUS	X	X																					X												X	X						X	X		
PRIZER	X	X																							X																				
BRANCH	X	X							X											X			X						X	X					X	X						X	X		
SEPARATR	X	X																					X												X	X						X	X		
Black box	X	X																																								X			
GE																																													
JETMIXER	X	X																				X																							
TURBINE	X	X																																							X				
FWHTR	X	X													X																														
ECCMIX	X	X								X																																			
VALVE	X	X													X								X				X								X	X						X	X	X	
CHKVLV	X	X																																										X	
TRPVLV	X	X																					X												X	X						X	X	X	
INRVLV	X	X																									X																	X	
MTRVLV	X	X																					X												X	X						X	X	X	
SRVVLV	X	X													X																													X	
RLFVLV																																													
PUMP	X	X	X																				X												X	X						X	X		
CPRSSR	X	X																											X																
MTPLJUN	X	X		X																								X					X	X											
ACCUM	X	X																					X	X											X	X						X	X		
MULTID	X	X		X																								X					X	X											
SNGLFW																																													
MTPLFW																																													

The Features Tests Matrix organizes and displays the features tested by individual input decks. With exceptions in columns 1–3, the columns correspond to input decks and the rows to single features. An "X" means the feature in the corresponding row is tested by the input deck named at the top of the column. An "X" in column two indicates the feature is tested by at least one member of the suite of cases, while an "X" in column three indicates that it is restarted.

Features that are not tested have blanks in columns 2 and 3 and are marked in purple text, as shown in Figure 2. With over 9200 cells in the table, manual construction requires a great deal of time and checking, so large portions of the construction are automated. To aid matrix construction, RELAP5-3D writes helpful information for building the matrix on the printed output file when verification is active. Even so, sections manually prepared may not mark every input deck that tests a given feature. Automation becomes more important as addition of new input and code features requires the matrix to be rebuilt.

The verification tests are summarized in Table 2. Each test resides in a directory named in column 1 that contains input and auxiliary files. The base input file has the same name as the directory, but with a ".i" extension. The verification file will have the ".vrf" extension.

Table 2 Verification Tests and Descriptions

Test ID (# Cases)	Description
3dflow (18)	Simulates 3-D flow of single-phase liquid, single-phase vapor, or two-phase flow in a 3x3x3 Cartesian grid with either 1-D or 3-D momentum equations.
ans (9)	Tests decay heat options with the point kinetics model, fission power types, fission product types available with each ANS standard, and the G-factor contribution to the decay heat.
boronm (4)	Tracks a square wave in boron concentration through a constant area pipe with and without Godunov method.
crit (4)	Tests Ransom-Trapp and Henry-Fauske critical flow models for a range of stagnation conditions including subcooled, two-phase, and superheated in a small horizontal pipe. Also tests cases with no choking allowed and homogeneous flow.
cyl3 (1)	Tests the metal water reaction model for steam flowing past the right surface of a cylindrical heat structure.
duklerm (5)	Tests the CCFL model using Dukler-Smith air-water countercurrent flow data. Wallis, Kutateladze, and Bankoff correlations are tested.
eccmix (1)	Models a portion of the cold leg of a typical PWR during ECC injection.
edhtrkm (5)	Edward's pipe simulates a rapid blowdown of a pipe. Includes extras: reactor kinetics, heat structure cosine temperature problems, and all control variables types, but shaft. Cases use fluids: h2o, d2o, h2on, h2o95, hen, and an air/water mixture.
eflag (2)	Simulates blowdown of one vessel into another to check the effect of the e-flag on the thermodynamic state in the downstream vessel.
enclss (1)	Steady-state calculation of a graphite stack using the heat conduction enclosure model.
fric (14)	Tests various single-phase wall and junction friction models. Cases include turbulent flow with and without heated wall effect, laminar flow with and without shape factors, abrupt area change options, and user input equations for wall and form friction.
fwhtr (1)	Represents a tube-in-shell feedwater heater.
gota27 (1)	Simulates rod-to-rod radiation in a 64-rod bundle in low-pressure steam using radiation enclosure model.
hse (3)	Simulates two-phase flow through a horizontal tee with offtakes coming off the top, bottom, or side face of the horizontal pipe.
httable (3)	Simple model of a pipe and heat structure exercising structure BC related to heat flux and heat transfer coefficient.
httest (9)	Simple model of a pipe and heat structure that varies IC and BC to achieve various heat transfer regimes for heat transfer packages 1, 111, and 134. Also tests the non-equilibrium volume option.
hxco2m (2)	Models a once-through heat exchanger with PbBi on the shell side and supercritical carbon dioxide inside the tubes. Tests the normal and alternate heat structure-fluid coupling models in steady-state.
jetjun (2)	Simulates insurges and outsurges of liquid into a pressurizer with and without the jet junction model.
jetpmp (1)	Tests jet pump performance over a range of suction and driveline flows.
l31acc (1)	Represents the accumulator response during a slow depressurization during LOFT Experiment L3-1.
l2-5-emA (1)	Tests Appendix K options during a LOFT Experiment L2-5, which simulates a loss-of-coolant accident initiated by a large break.
neptunus (2)	Models pressurizer insurge/outsurge experiment with spray.
pack (4)	Vertical fill problem tests water packing model when subcooled liquid is injected into superheated steam from below. Uses semi- and nearly-implicit timesteps.
pitch (1)	Tests an inertial check valve with movement.
radial (1)	Models pure radial, symmetric flow problem in a 2D hollow cylinder. There is no azimuthal flow.
rcpr (1)	Tests the performance of a recompressing compressor in a supercritical CO2 cycle.
refbun (1)	Tests two-phase flow and heat transfer with horizontal and vertical bundles that exercise the Groeneveld and PG CHF correlations and correlations for narrow, rectangular channels.
regime (22)	Tests the standard horizontal and vertical flow regimes by adjusting flow boundary conditions through a simple pipe. Both the pre-CHF and post-CHF regimes are tested for the vertical pipe.
rigidbody (1)	Models pure azimuthal, symmetric flow problem in a 2D hollow cylinder. There is no radial flow.
rtheta (1)	Models flow in a 2D hollow cylinder with symmetric flow in both the radial and azimuthal flow directions.
rtsampnm (1)	Based on typpwr, tests radio-nuclide transport model and the axial heat source options using nodal kinetics.
rtsamppm (1)	Based on typpwr with uses point kinetics, tests various axial heat source options, including those from tables, control variables, and reactor kinetics. Tests the radio-nuclide transport model too
slab3 (1)	Tests the metal water reaction model for steam flowing past the right surface of a rectangular heat structure.
sphere3 (1)	Tests the metal water reaction model for steam flowing past the right surface of a spherical heat structure.
state (24)	Tests various fluid states, including subcooled liquid, two-phase, superheated vapor, high-pressure liquid, high-temperature vapor, and supercritical, for h2o, h2on, d2o, and new helium.
todcnd (1)	Models heat transfer from hot wall with the reflood and two-dimensional heat conduction models.
turbine9 (1)	Multi-stage steam turbine with moisture separation. All four types of turbines are tested.
typ1200 (1)	Models small-break LOCA in a typical pressurized water reactor for 1200 s.
typ_kindt (2)	TYPPWR input model with nodal kinetics, Krylov solver, and independent kinetics timestep.
valve (5)	Models opening and closing of all valves, except relief.
varvol2 (1)	Uses the variable volume model and a general table to vary the fluid volume of a single liquid-filled volume.
2phspump (1)	Tests two-phase pump head degradation as a function of void fraction alone and as a function of void fraction and pressure.

The number of input cases in the base input deck is listed after each test name in Table 2. All kernels are recorded in the same verification file with the cases marked as shown in Fig. 1. The tests run in 1 to 10 seconds on modern computer platforms with 3 exceptions that take under one minute each.

NULL TESTING

There are many times in the development of codes such as RELAP5-3D that it is desired to show that code modifications do not impact results. Examples of this are situations when memory structure has been modified, or ensuring that new features do not change the results of problems in which they are not employed. The combination of the verification data files and a comprehensive test suite provide the ability to test this condition. To perform null testing, a reference set of solutions is first generated. The test suite is then executed with the modified code. If all of the verification files are the same, there is now evidence that the changes have not inadvertently impacted the results.

RESTART TESTING

The RELAP5-3D restart capability[6] provides a means to continue a previous calculation from some point after input processing concluded, possibly from the end of the previous run or an intermediate point in the calculation. Each restart record contains virtually all calculation parameters (E.G. pressures, temperatures, void fractions, flow rates, etc.) for the entire transient calculation.

Previously, users have reported that for some input models, the code produces different calculations on restart than it does when the code runs straight through. Acceptable restart performance for an input deck means that a restart run from an intermediate record of its restart file produces the same verification kernels as the base run did, for all common timesteps.

The purpose of restart testing is to ensure this for the tested features. All 43 tests of the verification suite are restarted and the restart input files are stored with the base input deck. If the base deck is named base.i, then the restart is named base.r.i and the verification file is base.r.vrf. If more than one input case exists, then the restart files it produces are named base_1.r, base_2.r, etc. and base.r.i uses them to run the multiple restart cases[6], but all verification kernels are written to base.r.vrf.

Like with null testing (version-to-version comparison) all identification data is ignored when comparing base and restart verification files. The test fails for a given input model if differences are found. The process ensures that all verification kernels occur after the restart time, to prevent false differences from being found.

Table 3 Issues Found and Fixed via New Verification

Description	Found	Fixed
Restart issues	10	5
Restart issue w/ final bit of cumulative time	(5)	(0)
Backup issues	27	17
Total Issues	**37**	**22**

The new verification capability revealed 10 restart issues. Five have been resolved. The remaining five have been tracked to a difference in the final bit of cumulative time between the restart file and internal memory. Table 3 summarizes this.

BACKUP TESTING

A RELAP5-3D time-step backup occurs when the code detects and improper solution has been obtained. For these conditions, the solution is reset to the values at the end of the last successful advancement and the time-step is repeated, often with a modified logic path, with either the same or reduced time-step size.

Backup testing checks that the code can correctly repeat any given advancement at the same time-step size. The process artificially induces a backup, as directed from code input, by setting set one of the code's three backup condition[7, 8, 9] flags on one or more advancements. The process ensures that the same time-step size is used on the repeated time-step. The code must produce the same calculations on both attempted advancements. Since this capability is used in all applications, it is important to ensure that the code results are not impacted through imperfect backup logic.

To examine the backup capability, two runs are made. One uses the original deck of a given input model and the second uses a copy of that deck with forced backup. However, not all advancements stress the code equally. Which one is the best for revealing backup errors?

Since code updates can cause some phenomena to occur at different times, or to take a different number of advancements to reach the same point in time, constant resetting of the backup time would be required for some input models as the code is developed. A better solution is to force a backup on every successful advancement (except the first because it lacks certain old-time values that must be restored on a backup) to create a thorough and non-changing backup test. This can be invoked via input with the keyword "backall".

Backall produces a verification kernel only on the final advancement, otherwise the verification file would reach the 1MB limit for many tests.

Two copies of the base input deck, base.i, are made for backup testing. The first, base.b.i, performs the base run but writes a kernel only for the final advancement. The second, base.bk.i, runs with the backall option. Building these decks from base.i every time backup testing is performed ensures that the base and backup decks match perfectly, even if the base deck is modified. Before comparing the verification files, base.b.vrf and base.bk.vrf, the line with repeat counts is removed because the forced backups change them artificially.

The backup testing has indentified that 27 of the verification problems did not pass the rigorous backup testing, see Table 3. Of these, 17 have been resolved. Work is ongoing to resolve the final 10.

PROGRAMMING AND AUTOMATION

The verification coding is largely separated from the rest of the source code in RELAP5-3D. Most of the verification

subprograms, along with their new data and documentation, are stored in a new module. This isolates the new coding; thus it reduces the chance of introducing errors when adding new code. Very few statements need to be inserted into the existing RELAP5-3D code to access this data and these routines. This approach makes it is easier to port the verification coding to older versions of RELAP5-3D and would aid in its adaptation to other Fortran programs.

However, not all the new coding is in the module. In some existing subroutines, such as the 199-card processor, internal subprograms that provide verification-specific capability were written. Also, two new standalone subroutines were written. These routines access data of several different modules so that including them in the new verification module would subvert the goal if isolating new code.

The method is invoked through a system of Makefiles on either a Linux platform or a properly equipped Windows PC. The user must first identify the locations of the RELAP5-3D distribution, the verification test suite, and directory for the output verification files in an auxiliary file. The user then issues a command that invokes the Makefile with one of four major options: null test, restart test, backup test, or all tests.

The Makefile runs all requested tests, put the verification files in the target directory, and compares them to previous verification files if they exist. If everything compares exactly, the Makefile reports the single word: verified. If there are differences, it reports that and lists the test problems that did not compare. If all tests are run, it categorizes the failures in the report. This system was used in creating Table 3.

The automated system employs two Makefiles. They perform the tasks explained above and provide other functions. For example, the system builds the backup input files, links fluid property files and RELAP5-3D executable program to each test directory, renames existing verification files at the target location as backup files with the 'bak' extension before comparison, and performs a variety of cleaning operations.

STATISTICAL ANAYLSIS

The verification testing developed above is a form of hypothesis testing and can be analyzed statistically. The null hypothesis of the test, H_0, is: "The two runs produce exactly the same calculations." The alternate hypothesis is that "the calculations are different." The statistic used to test the hypothesis for the i^{th} test case, X_i, has a value of 0 if no differences are found between the two runs. It is 1 otherwise, no matter how many differences occur. When there are N test cases, then X_i is defined for each test case, i, and X is the maximum. Applying standard statistical methods[10], this can be expressed mathematically as:

$$X_i = \begin{cases} 0 & \text{if exactly 0 differences are found} \\ 1 & \text{if at least one difference is found} \end{cases} \quad (1.1.1)$$

$$X = \max \{X_i \mid i = 1, 2, \ldots, N\} \quad (1.1.2)$$

H_0: The two runs have identical calculations

Test V: Accept H_0 if X = 0, but reject it if X = 1.

Hypothesis testing potentially commits two kinds of errors. Type I Error or false positive, is the rejection of H_0 when it is true, in other words, finding a difference when there are none. The probability of Type I Error is denoted α and is called the level of significance of the test. Type II Error or a false negative, is accepting H_0 when it is false. That means there are differences that go undiscovered. It has probability β.

$\alpha = P(\text{Reject } H_0 \mid H_0 \text{ is true}) = P(\text{Type I error})$

$\beta = P(\text{Accept } H_0 \mid H_0 \text{ is false}) = P(\text{Type II error})$

This is summarized in Table 4.

Table 4 Hypothesis Testing Table

	H_0 is true No differences exist	**A_0 is true** Differences exist
Accept H_0	*Correct* Report: "Verified"	*Type II Error* Fail to find actual differences
Reject H_0	*Type I Error* Falsely find differences	*Correct* Report: "Differences have been found"

Analysis shows that Test V has good properties.

Theorem: Test V always accepts the null hypothesis when it is true.
Proof: Suppose H_0 is true. There are no differences to detect, so for each test in the suite, $X_i = 0$. Therefore,
$X = \max \{X_i\} = 0$ *always* (when H_0 is true). Thus,
$P(\text{Accept } H_0 \mid H_0 \text{ is true}) = P(X = 0 \mid H_0 \text{ is true}) = 1$.

This only applies if the test is properly programmed.

Corollary: The test never commits Type I Error. It has significance level, $\alpha = 0$.

Note that these two results apply to all three kinds of testing presented here: null, restart, and backup.

Regarding Type II error, β is less than 10^{-32} for primary and secondary variables for any single test problem, but Test V may miss errors in output-only quantities. Used in conjunction with the diffem utility described in the introduction, the combined method detects all differences to the accuracy of the printed output file, so β is approximately 10^{-5}. Since the power of a statistical test is 1-β, the combined method is a very powerful test of a single input problem.

SUMMARY

A highly accurate, automated testing system has been implemented for RELAP5-3D verification. It employs a set of input problems, called the verification test suite, that cover the important code features for nuclear power plant modeling. It writes a small, extremely accurate verification file of L_1 norms of primary variables to detect differences in the first 32 significant digits. It can be used for null testing and extends the original scope of verification to restart and backup testing. The method is very powerful and can be applied to many other computer programs.

REFERENCES

1. The RELAP5-3D Code Development Team, "RELAP5-3D Code Manual Volume I: Code Structure, System Models and Solution Methods," INL-EXT-98-00834-V1, Revision 4.0, Section 8.2, p 8-4, June, 2012.
2. D. L. Aumiller, et al, "Development of Verification Testing Capabilities for Safety Codes," The 15th International Topical Meeting on Nuclear Reactor Thermal - Hydraulics, NURETH-15, Pisa, Italy, May 12-17, 2013.
3. The RELAP5-3D Code Development Team, "RELAP5-3D Code Manual Volume III: Developmental Assessment," INL-EXT-98-00834-V1, Revision 4.0, Section 8.2, p 8-4, June, 2012.
4. W. Kahan "IEEE Standard 754 for Binary Floating-Point Arithmetic," Elect. Eng. & Computer Science, Univ. of California, Berkeley CA 94720-1776, p4, October, 1997.
5. G. L. Mesina, "Reformulation RELAP5-3D in FORTRAN 95 and Results," Proceedings of the ASME 2010 Joint US-European Fluids Engineering Summer Meeting and 8th International Conference on Nanochannels Microchannels, and Minichannels, FEDSM2010-ICNMM2010, Montreal, Quebec, Canada, Aug 1-5, 2010.
6. The RELAP5-3D Code Development Team, RELAP5-3D Code Manual Volume V: User's Guidelines, INEEL-EXT-98-00834, Revision 4.0, Section 4.1.2, pp. 4-2, Idaho National Laboratory, PO Box 1625, Idaho Falls, Idaho 83415, June, 2012.
7. The RELAP5-3D Code Development Team, RELAP5-3D Code Manual Volume II: User's Guide and Input Requirements, INEEL-EXT-98-00834, Revision 4.0, Section 8.7, pp. 8-31 to 8-34, Idaho National Laboratory, PO Box 1625, Idaho Falls, Idaho 83415, June, 2012.
8. The RELAP5-3D Code Development Team, "RELAP5-3D Code Manual Volume I: Code Structure, System Models and Solution Methods," INL-EXT-98-00834-V1, Revision 4.0, Section 8.2, pp. 8-3 to 8-4, June, 2012.
9. The RELAP5-3D Code Development Team, "RELAP5-3D Code Manual Volume I: Code Structure, System Models and Solution Methods," INL-EXT-98-00834-V1, Revision 4.0, Section 3.4, pp. 3-271 to 3-274, June, 2012.
10. V. K Rohatgi, An Introduction to Probability Theory and Mathematical Statistics, Second Edition, ISBN-10: 0-471-34846-5, John Wiley & Sons, Inc., NY, Oct 2000.