DFTCalc

Calculating DFTs using Lotos NT

Author: Van der Berg, F.I.

Date: May 25, 2012

University of Twente

Contents

1 Introduction

During the design of a mission-critical component-based system one has to take failures into account. One way to model the failure of a component-based system is by using Dynamic Fault Trees (DFT). A DFT describes the dependencies (edges) the components (nodes) have on each other on multiple levels. Each leaf-node describes a basic component or event and other nodes describe part of the system comprised of one or more basic components or events. Which this knowledge, the failure rate of the whole system can be calculated.

Figure 1 (Galileo format in Figure 2) illustrates a small example of a 2-disk RAID1 array[1] modeled as a DFT. The basic components are the two disks. The RAID fails when both disks fail, hence the **And** node. The **And** node describes the RAID part of the system. This example could be expanded to two RAIDs, joined together with for example an **Or** node. The **Or** node would specify that the whole system fails if either or both of the RAIDs fail.

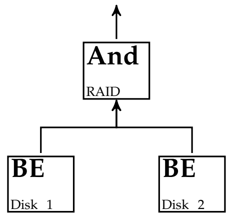

Figure 1: RAID1 array[1] as an **And** node and **Basic Event** nodes

raid.dft

```
toplevel "RAID";
"RAID" and "Disk1" "Disk2";
"Disk1" lambda=0.01;
"Disk2" lambda=0.01;
```

Figure 2: The RAID1 DFT in Galileo format

1.1 Coral

Boudali, Crouzen and Stoelinga show how the failure probability of a DFT can be calculated using I/O-IMCs. [BCS07] Their approach consists of three stages. The first stage translates each node in the DFT to the process algebra *Lotos* (Language Of Temporal Ordering Specification) [BB87], which is then compiled to an IMC . In stage two these separate IMCs are then parallel composed together into one IMC. In the final stage the failure probability of the DFT is calculated using this IMC. They implemented their approach in the tool *Coral*.

FIXME io imc or imc?

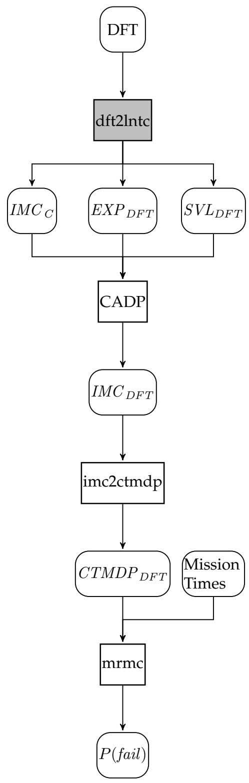

Figure 3: The dftcalc toolchain, ▭ means new

1.2 Lotos NT

Coral translates nodes into the process algebra Lotos. While this language is very expressive, it is not the most readable language. This is one of the rationale behind *Lotos NT*: to make the language more readable. [Sig08]

Lotos NT defines a clean syntax in which modules can be defined. These modules may be thought of as being similar to Java packages: they group functionality and modules may depend on other modules. In these modules, processes, function, variables and so on may be defined. Lotos NT is used to describe

EXP and *SVL* are two other languages that are used. EXP is a language to denote a network of processes. It is used to describe how the individual components of the DFT can be glued together to represent the actual DFT. The Script Verification Language (SVL) is used to describe and execute the tasks of generating individual components and the glueing together.

The rest of this paper will be layed out as follows. In section 2 we will introduce the new *DFTCalc* tools and how they fit in the existing tool chains. In section 3 the tool is compared to existing tools and in section 4 we conclude this paper.

[1] Redundant Array of Independent Disks, level 1 (mirroring)

2 DFTCalc

DFTCalc serves as the successor to Coral. It shares the same goal as Coral: to calculate the failure probability of a DFT. The main differences between the two are that 1) DFTCalc uses the newer Lotos NT to describe the building blocks; 2) DFTCalc generates EXP to glue the building blocks together; 3) DFTCalc is built to support future dynamic additions such as repair rates.

The tool is separated in three distinct tools:

1. **dft2lntc** handles the generation and conversion of individual nodes in the DFT to IMCs. It also generates the script that glues the components together to form the IMC representing the specified DFT. It does not create this IMC however.
2. **dftcalc** handles the glueing of individual nodes (IMCs) together to form the IMC that represents the DFT, and handles the calculation of the resulting IMC. All tasks are out sourced to other tools, including dft2lntc. See Figure 3 for an overfiew of the dftcalc tool.
3. **dfttest** provides a test suite handler for DFTs with multiple backends (e.g. coral, dftcalc) and keeps track of all statistics like resources (time, memory) and resulting faulure rate.

2.1 dft2lntc

The tool dft2lntc handles the generation and conversion of individual nodes in the DFT to IMCs. As input, one specifies the DFT of which the individual nodes are to be transformed into Lotos NT and IMC components. It then checks a global archive (e.g. in `/share/dft2lnt`) for the IMC representing this node. If it is not there, it will generate this and put it in the global archive. This avoid recalculating each component every time any DFT is calculated.

The tool also generates the script that glues the components together to form the IMC representing the specified DFT. This is done by generating an EXP script that specifies how each component should communicate with the other components. Smart composition will determine the optimal order of composition and apply it. The composition is performed when the generated SVL script is called. The SVL script is called by dftcalc. Figure 4 shows the generated EXP and SVL files for the RAID example.

FIXME optimal?

2.2 dftcalc

The tool dftcalc performs the actual calculation of the failure probability of the fault tree. It first calls dft2ltnc to make sure individual components are available as their IMC counterpart and to generate the SVL glue script. It then calls this SVL glue script to create the IMC representing the DFT. This IMC is converted to a CTMDP (Continuous time Markov decision process), which is fed to mrmc together with the mission time. The output of mrmc is the failure probability.

Figure 4 Generated **raid.exp** and **raid.svl** by dft2lntc

raid.exp

```
(* Number of rules: 6*)
hide
    a_and0_be1,
    a_and0_be2,
    f_be1,
    f_be2
in
    label par
    (*  and0                   be1                    be2                    *)
        "ACTIVATE !0 !FALSE" * _                    * _                    -> ACTIVATE,
        "ACTIVATE !1 !TRUE"  * "ACTIVATE !0 !FALSE" * _                    -> a_and0_be1,
        "ACTIVATE !2 !TRUE"  * _                    * "ACTIVATE !0 !FALSE" -> a_and0_be2,
        "FAIL !0"            * _                    * _                    -> FAIL,
        "FAIL !1"            * "FAIL !0"            * _                    -> f_be1,
        "FAIL !2"            * _                    * "FAIL !0"            -> f_be2
    in
        "and_p1_c2.bcg"
        ||
        total rename "RATE_FAIL !1 !2" -> "rate 0.01" in "be_p1_cold.bcg" end rename
        ||
        total rename "RATE_FAIL !1 !2" -> "rate 0.01" in "be_p1_cold.bcg" end rename
    end par
end hide
```

raid.svl

```
"raid.bcg" = smart stochastic branching
             reduction of "raid.exp"
```

how to generate:

```
$ dft2lntc raid.dft -oraid
```

The entire process is logged and statistics are kept. In the event of an error in any of the links in the tool chain, the error log is presented. The resulting output is written in YAML[5] format to a specified file or to stdout. Figure 5 shows an example output in YAML format.

raid.result

```
raid.dft:
  dft: raid.dft
  failprob: 9.9e-05
```

Figure 5: Result of dftcalc in YAML[5] format, containing one DFT calculation

2.3 dfttest

The tool dfttest provides a test suite handler for DFTs with multiple backends (e.g. coral, dftcalc). It gives the ability to execute one or more DFTs specified in the test suite. All data of every execution is kept in the test suite file. Data includes failure probability, statistics about the generated IMC (states, transitions), and resource statistics (time, memory). Tests in the test suite that have been done are not executed again but their result is cached. Running these tests can be forced if desired. Forcing the tool to only output cached results is also supported. Figure 7 shows an example output when dfttest is run with **raid.test** in Figure 6 as input.

The timing of the executions are done by using hardware timers. This avoids inaccuracies caused by programs altering the system clock, such as NTP. On Unix-like OSes this is achieved by using `CLOCK_MONOTONIC_RAW` and on Windows by using `QueryPerformanceCounter`. Memory statistics are given by SVL when called to generate the IMC representing the DFT.

raid.test

```
- general:
    fullname: RAID1 Array, two disks
  dft: /opt/dftroot/raid.dft
  timeunits: 1
  verified:
    manual:
      failprob: 9.9e-05
  results:
    - 2012-03-17 20:35:16:
        coral:
          stats:
            time_monraw: 15.4715
          failprob: 9.9e-05
          bcginfo:
            states: 6
            transitions: 11
        dftcalc:
          stats:
            time_monraw: 3.93587
          failprob: 9.9e-05
          bcginfo:
            states: 5
            transitions: 8
```

Figure 6: Test suite in YAML format, containing one test

2.4 Package

The tool set can be downloaded from its Git[1] repository[2]. Bug reports, feature requests, comments or general praises can be proclaimed on the project management site[3].

Building the tool suite is a matter of running `cmake` with your desired generator (only makefiles were tested) and then executing the build process. Under Windows, MSYS[4] can be used to build makefiles. The package includes the source of all library dependencies (YAML[5]) and should be buildable provided the following tools are present on the system:

- **CMake**, v2.8+, tested with v2.8.7
- **GNU Make**, tested with v3.82
- **GCC**, v4.6+, tested with v4.6.{1,2,3}
- **GNU Bison**, tested with 2.5.35
- **Flex**, tested with v2.5

Note that to actually perform calculations you need **mrmc**, **Coral** and **CADP**.

MISSING download location

MISSING redmine location

[1] Git is a free & open source, distributed version control system, `http://git-scm.com/`
[2] DFTCalc's Git repository, `http://...`
[3] DFTCalc's project management site, `http://...`
[4] MSYS is a collection of GNU utilities, it is intended to supplement MinGW, `http://www.mingw.org/wiki/MSYS`
[5] YAML Ain't Markup Language, `http://yaml.org/`

Figure 7 Running dfttest on **raid.test**

raid.exp

```
$ dfttest raid
::  Test RAID1 Array, two disks
    Iteration     ¦ Time (s)      ¦ Memory (MiB) ¦ P(fail)       ¦ States        ¦ Transitions   ¦ Speedup
    manual        ¦             - ¦            - ¦ 9.9e-05       ¦             - ¦             - ¦ -
 o  coral         ¦        15.428 ¦       13.477 ¦ 9.9e-05       ¦             6 ¦            11 ¦ 1
 o  dftcalc       ¦         4.210 ¦       16.242 ¦ 9.9e-05       ¦             5 ¦             8 ¦ 3.66456
 Test OK
```

3 Results

Figure 8 shows results for the same dynamic fault trees used in [BNS09] and [BCS07]. They are included in the git repository. The DFT `ftpp_weibull` was not tested because, like Coral, DFTCalc does not support the Weibull distribution.

The tests were performed in a virtual machine, running on two cores of an *AMD Phenom(tm) II X6 1090T Processor*. The maximum amount of memory available was *2GB*.

The results show that DFTCalc is a significant improvement performance-wise over Coral. It is roughly twice as fast. The memory difference could not be measured as Coral does not seem to lend itself for this. From manual monitoring the processes it seemed that DFTCalc peaks at about twice as much memory as Coral.

The reason for this speed up can be attributed to a few things. First of all the caching of IMCs of individual nodes speeds up the process if the IMC for a node has already been built earlier. Secondly DFTCalc directly generates an EXP file, without first generating an SVL file which generates the EXP file. This probably yields only a minor increase. Lastly a major difference is the use of smart composition. This optimizes the composition of the individual IMCs to the complete IMC representing the DFT.

Figure 8 Results

	Iteration	Time (s)	$\text{Memory}_{\text{peak}}$ (MiB)	P(fail)	States	Transitions	Speedup
cps	coral	102.988	15.027	0.0013567	39	108	1
	dftcalc	44.571	13.348	0.0013567	39	108	2.31066
cas	coral	161.902	17.676	0.6579	16	50	1
	dftcalc	57.809	13.348	0.6579	30	106	2.80064
mdcs	coral	134.829	16.449	0.0666448	22	69	1
	dftcalc	51.393	16.242	0.0666448	28	86	2.62347
ftpp_standard	coral	613.531	40.461	0.0192186	142	923	1
	dftcalc	200.355	220.309	0.0192186	72	386	3.06222
ftpp_large	coral	881.926	54.414	0.0030616	2167	27438	1
	dftcalc	505.156	490.953	0.0030616	400	3369	1.74585
ftpp_complex	coral	-	-	-	-	-	-
	dftcalc	1527.950	598.441	0.0213576	20750	339718	-

4 Conclusions

We have introduced a new tool: DFTCalc. This tool is meant as the successor to Coral and we have shown that the new tool is about twice as fast as Coral. By using Lotos NT as the language to model individual nodes we obtain clean code, without sacrificing in expressiveness.

The current implementations of *dft2lntc* and *dftcalc* are easily adaptable for future extensions, such as repair rates. The code for the individual nodes was implemented with exactly this in mind. The tool itself is implemented in well-documented C++, providing both extensibility and optimized binaries. The test suite manager *dfttest* provides an easy way to manage test specifications and test results.

The next step for this tool is to add repair rates to *dft2lntc*, by extending the implementation of the individual nodes and the EXP glue-code. For *dftcalc* the next step is to add more options for the user to specify what is to be calculated, e.g. averages and evidence[1] (specifying that certain Basic Events fail right at the start). The test suite manager *dfttest* can be improved by parameterizing the test specifications and specifying the concrete values in concrete results. For example a list of mission times could be specified and for each mission time a different concrete result is calculated.

The tool can be freely downloaded and executed from []. You also need mrmc and Coral binaries as well as a licensed installation of CADP.

MISSING download location

[1] In the most recent implementation evidence is implemented

References

[BB87] Bolognesi, T. and E. Brinksma
Introduction to the ISO specification language LOTOS. *Comput. Netw. ISDN Syst.*, 14(1):25–59, March 1987.

[BCS07] Boudali, H., P. Crouzen, and M. I. A. Stoelinga
A compositional semantics for Dynamic Fault Trees in terms of Interactive Markov Chains. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07), Tokyo, Japan*, volume 4762 of *Lecture Notes in Computer Science*, pages 441–456, Berlin/Heidelberg, October 2007. Springer.

[BNS09] Boudali, H., A. P. Nijmeijer, and M. Stoelinga
DFTSim: a simulation tool for extended dynamic fault trees. In *SpringSim*, 2009.

[Sig08] Sighireanu, M.
LOTOS NT User's Manual (Version 2.6). `http://vasy.inria.fr/traian/manual.html`. INRIA projet VASY, February 2008.

A Appendix A

Help output of the three tools.

A.1 dft2lntc

```
:: dft2lntc [INPUTFILE.dft] [options]
  Compiles the inputfile to EXP and SVL script. If no inputfile was specified,
  stdin is used. If no outputfile was specified, 'a.svl' and 'a.exp' are used.

:: General Options:
  -h, --help       Show this help.
  --color          Use colored messages.
  --no-color       Do not use colored messages.
  --version        Print version info and quit.

:: Debug Options:
  -a FILE          Output AST to file. '-' for stdout.
  -t FILE          Output DFT to file. '-' for stdout.
  --verbose=x      Set verbosity to x, -1 <= x <= 5.
  -v, --verbose    Increase verbosity. Up to 5 levels.
  -q               Decrease verbosity.

:: Output Options:
  -o FILE          Output EXP to <FILE>.exp and SVL to <FILE>.svl.
  -x FILE          Output EXP to file. '-' for stdout. Overrules -o.
  -s FILE          Output SVL to file. '-' for stdout. Overrules -o.
  -b FILE          Output of SVL to this BCG file. Overrules -o.
  -e evidence      Comma separated list of BE names that fail at startup.
  --warn-code      Return non-zero if there are one or more warnings.
```

A.2 dftcalc

```
:: dftcalc [INPUTFILE.dft] [options]
  Calculates the failure probability for the specified DFT file, given the
  specified time constraints. Result is written to the specified output file.
  Check dftcalc --help=output for more details regarding the output.

:: General Options:
  -h, --help       Show this help.
  --color          Use colored messages.
  --no-color       Do not use colored messages.
  --version        Print version info and quit.
  -O<s>=<v>        Sets settings <s> to value <v>. (see --help=settings)

:: Debug Options:
  --verbose=x      Set verbosity to x, -1 <= x <= 5.
  -v, --verbose    Increase verbosity. Up to 5 levels.
  -q               Decrease verbosity.

:: Output Options:
  -r FILE          Output result to this file. (see --help=output)
  -p               Print result to stdout.
  -t x             Calculate P(DFT fails in x time units), default is 1
  -m <command>     Raw MRMC Calculation command. Overrules -t.
  -C DIR           Temporary output files will be in this directory

:: Settings
  Use the format -Ok=v,k=v,k=v or specify multiple -O

:: Output
  The output file specified with -r uses YAML syntax.
  The top node is a map, containing one element, a mapping containing various
  information regarding the DFT. E.g. it looks like this:
  b.dft:
    dft: b.dft
    failprob: 0.3934693
    stats:
      time_user: 0.54
      time_system: 0.21
      time_elapsed: 1.8
      mem_virtual: 13668
      mem_resident: 1752
  The MRMC Calculation command can be manually set using -m. The default is:
    P{>1} [ tt U[0,x] reach ]
  where x is the specified number of time units using -t, default is 1.
```

A.3 dfttest

```
:: dfttest [options] [suite.test]
  Calculates the failure probability for the DFT files in the specified test
  file. Result is written to stdout and saved in the test file.
  Check dfttest --help=input for more details regarding the suite file format.
  Check dfttest --help=output for more details regarding the output.

  If the specified suite file does not exist, it will be created. If the suite
  file is not writable, you will be asked to specify a suite file to save to.
  If that suite file already exists, the suites will be merged in such a way
  that nothing is overwritten.

:: Common usage:
  dfttest <suite.test> -t <tree.dft>   Run only <tree.dft>, adds <tree.dft>
  dfttest <suite.test> -ct <tree.dft>  Adds <tree.dft>, no test is performed

:: General Options:
  -h, --help       Show this help.
  --help=x         Show help about topic x.
  --color          Use colored messages.
  --no-color       Do not use colored messages.
  --version        Print version info and quit.
  -O<s>=<v>        Sets settings <s> to value <v>. (see --help=settings)

:: Debug Options:
  --verbose=x      Set verbosity to x, -1 <= x <= 5.
  -v, --verbose    Increase verbosity. Up to 5 levels.
  -q               Decrease verbosity.

:: Test Options:
  -c               Do not run tests, only show cached results.
  -f               Force running all tests, regardless of cached results
  -t DFTFILE       Add/Limit testing to this DFT. Multiple allowed.
  -L               Output is the content of a LaTeX tabular. Implies -c.
  -C               Output is in CSV. Implies -c.

:: Help topics:
  input            Displays the input format of a suite file
  output           Shows some considerations about the output (timing)
  To view topics: dfttest --help=<topic>

:: Output
 > Timing
  Time measurements are done using platform specific implementations.
  On Linux, CLOCK_MONOTONIC_RAW is used and on Windows the API call
  QueryPerformanceCounter is used.
  Both implementations aim to assure there is no influence from other
  programs such as NTP. The measurement is as accurate as the clock of
  the hardware is.
 > Memory
  Memory measurements are done by SVL itself, using the program specified in
  CADP_TIME environment variable.
 > BCG Info
  Information of the generated BCG, like states and transitions, is obtained
  by calling bcg_info.

:: Suite Input
  A test suite file is a file in YAML format. It contains a list of tests, where
  each test is a map with settings. Supported keys in this map:
    - <key>       : <value>
    - general     : a map containing general information about the test:
      - fullname : a descriptive name of the test
      - longdesc : a longer description of the test
      - uuid     : a unique identifier for the test
      - format   : a unique identifier for the test
    - dft      : relative or absolute path to the DFT file
    - timeunits: result will reflect P("dft fails within timeunits")
    - verified : a map containing verified results as value and motives as key
    - results  : a list of maps containing resultmaps
                a resultmap's key is the time the test was started
                a resultmap's value is again a map with the obtained results as
                value and the origin of the results as key

  A complete example:
  - general:
      fullname: Basic Event
      uuid: 35896B7BE62877CD3255CA3E1579E976A2DDD9DDFEFC761A51075EBB97BD71A7
      longdesc: ""
      format: 1
    results:
      - 2012-02-29 17:36:11:
          coral:
            stats:
              time_monraw: 5.78413
            failprob: 0.3934693
            bcginfo:
              states: 4
              transitions: 7
```

```
          dftcalc:
            stats:
              time_monraw: 3.15412
              mem_virtual: 13668
              mem_resident: 1752
            failprob: 0.3934693
            bcginfo:
              states: 4
              transitions: 6
    verified:
      manual:
        stats:
          {}
        failprob: 0.3934693
        bcginfo:
          states: 0
          transitions: 0
    timeunits: 1
    dft: /opt/dftroot/b.dft

:: Settings
  Use the format -Ok=v,k=v,k=v or specify multiple -O
```