Coverage

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Prof Michael Quinn
From Slides by Dr. Niels
Burkhardt

(Acknowledgement: Kerstin Eder from the University of Bristol and Avi Ziv from the IBM Research Labs in Haifa have kindly permitted the re-use of their slides for educational purposes.)

CSCE 689 Advanced HW Verification Spring 2018

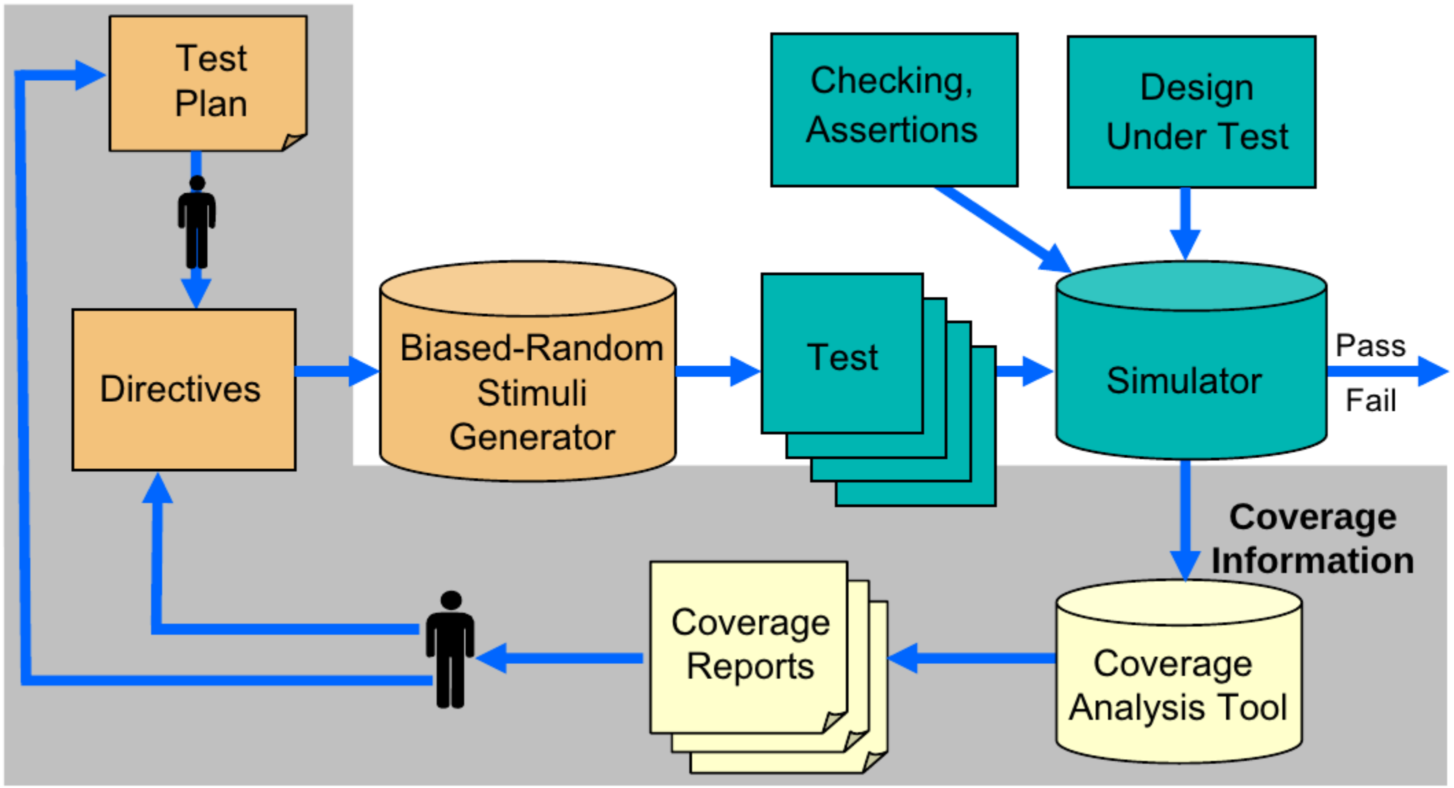
Outline

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Introduction to coverage
- Code coverage models
- Structural coverage models
- Functional coverage
- Coverage closure

Simulation-based Verification Environment

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386



Why Coverage?

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Simulation is based on limited execution samples
  - Cannot run all possible scenarios, but
  - Need to know that all (important) areas of the DUV are verified
- Solution: Coverage measurement and analysis
- The main ideas behind coverage
  - Features (of the specification and implementation) are identified
  - Coverage models written to quantify their behavioral spaces

Coverage Goals

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Measure the "quality" of a set of tests
- Supplement test specifications by pointing to untested areas
  - NOTE: Coverage gives ability to see what has not been verified!
  - Coverage completeness does not imply functional correctness of the design!
- Help create regression suites
  - Ensure that all parts of the DUV are covered by regression suite
- Provide a stopping criteria for unit testing
- Improve understanding of the design

Coverage Types

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Code coverage
- Structural coverage
- Functional coverage

- Other classifications
  - Implicit vs. explicit
  - Specification vs. implementation

Code Coverage - Basics

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Coverage models are based on the HDL code
  - Implicit, implementation coverage
- Coverage models are syntactic
  - Model definition is based on syntax and structure of the HDL
- Generic models – fit (almost) any programming language
  - Both software and hardware

Code Coverage - Limitations

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Code coverage can answer the question:

"Is there a piece of code that has not been verified?"

- Method used in software engineering for some time.

Main problem:

- False positive answers can look identical to true positive answers.

  False positive: A bad design is thought to be good.

- Useful for profiling:
  - Run coverage on testbench to indicate what areas are executed most often.
  - Gives insight on what to optimize!
- Many types of code coverage report metrics/models.

Types of Code Coverage Models

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Control flow
  - Check that the control flow of the program has been fully exercised
- Data flow
  - Models that look at the flow of data in, and between, programs
- Mutation
  - Models that check directly for common bugs

Control Flow Models

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Routine (function entry)
  - Each function / procedure is called
- Function call
  - Each function is called from every possible location
- Function return
  - Each return statement is executed
- **Statement (block)**
  - Each statement in the code is executed
- **Branch/Path**
  - Each branch in branching statement is taken
    - If, switch, case, when, …
- **Expression/Condition**
  - Each (sub-)expression in Boolean expression takes true and false values
- Loop
  - All possible number of iterations in (Bounded) loops are executed

Statement/Block Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Measures which lines (statements) have been executed by the verification suite.

```
✓ if (parity==ODD || parity==EVEN) begin
❑ parity_bit = compute_parity(data,parity);
  end
✓ else begin
✓ parity_bit = 1'b0;
  end
✓ #(delay_time);
✓ if (stop_bits==2) begin
✓ end_bits = 2'b11;
✓ #(delay_time);
  end
```

What do we need to do to get the statement coverage to 100%?

– Why has this never occurred?

– Is it a condition that can never occur? Was is simply forgotten?

– (Dead code can be “ok”!) WHY?

Path/Branch Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Measures all possible ways to execute a sequence of statements.

– Are all if/case branches taken?

– How many execution paths?

```
✓ if (parity==ODD || parity==EVEN) begin
✓ parity_bit = compute_parity(data,parity);
  end
✓ else begin
✓ parity_bit = 1'b0;
  end
✓ #(delay_time);
✓ if (stop_bits==2) begin
✓ end_bits = 2'b11;
✓ #(delay_time);
  end
```

☑ ☐ ☑ ☑

Note: 100% statement coverage but only 75% path coverage!

– **Dead code**: default branch on exhaustive case

– Don't measure coverage for code that was not meant to run! (tags)

Expression/Condition Coverage

Measures the various ways paths through the code are executed.

– Where a branch condition is made up of a Boolean expression, want to know which of the subexpressions have been covered.

```
✓ if (parity==ODD || parity==EVEN) begin
✓ parity_bit = compute_parity(data,parity);
  end
✓ else begin
✓ parity_bit = 1'b0;
  end
✓ #(delay_time);
✓ if (stop_bits==2) begin
✓ end_bits = 2'b11;
✓ #(delay_time);
  end
```

Note: Only 50% expression coverage!

- Analysis: Understand WHY part of an expression was not executed
- Reaching 100% expression coverage is extremely difficult.

Data Flow Models

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Coverage models that are based on flow of data during execution
- Each coverage task has two attributes
  - Define – where a value is assigned to a variable (signal, register, …)
  - Use – where the value is being used

```
process (a, b)
begin
   s <= a + b;
end process
process (clk)
begin
   if (reset)
      a <= 0; b <= 0;
   else
      a <= in1; b <= in2;
   end if
end process
```

Mutation Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Mutation coverage is designed to detect simple (typing) mistakes in the code
  - Wrong operator
    - + instead of –
    - >= instead of >
  - Wrong variable
  - Offset in loop boundaries
- A mutation is considered covered if we found a test that can distinguish between the mutation and the original code
  - Strong mutation – the difference is visible in the primary outputs
  - Weak mutation – the difference is visible inside the DUV

Code Coverage Models for Hardware

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Toggle coverage
  - Each (bit) signal changed its value from 0 to 1 and from 1 to 0
- All-values coverage
  - Each (multi-bit) signal got all possible values
  - Used only for signals with small number of values
    - For example, state variables of FSMs

Code Coverage Strategy

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Set minimum % of code coverage depending on available verification resources and importance of preventing post tape-out bugs.
  - A failure in low-level code may affect multiple high-level callers.
  - Hence, set a higher level of code coverage for unit testing than for system testing.
- Generally, 90% goal for statement, branch or expression coverage.
  - Some feel that less than 100% does not ensure quality.
  - Beware: Reaching full code coverage closure can cost a lot of effort!
  - This effort could be more wisely invested into other verification techniques.
- Avoid setting a goal lower than 80%.

Literature: [J Barkley. Why Statement Coverage Is Not Enough. A practical strategy for coverage closure., TransEDA.]

Structural Coverage

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Implicit coverage models that are based on common structures in the code
  – FSMs, Queues, Pipelines, …
- The structures are extracted automatically from the design and pre-defined coverage models are applied to them
- Users may refine the models
  – Define illegal events

State-Machine Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- State-machines are the essence of RTL design
- FSM coverage models are the most commonly used structural coverage models
- Types of models
  - State
  - Transition (or arc)
  - Path

Code Coverage - Limitations

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Coverage questions not answered by code coverage tools
  - Did every instruction take every exception?
  - Did two instructions accessed the register at the same time?
  - How many times did cache miss take more than 10 cycles?
  - Does the implementation cover the functionality specified?
  - …(and many more)
- Code coverage indicates how thoroughly the test suite exercises the source code!
  - Can be used to identify outstanding corner cases
- Code coverage lets you know if you are not done!
  - It does not indicate anything about the functional correctness of the code!
- 100% code coverage does not mean very much.
- Need another form of coverage!

Functional Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- It is important to cover the functionality of the DUV.
  - Most functional requirements can't easily be mapped into lines of code!
- Functional coverage models are designed to assure that various aspects of the functionality of the design are verified properly
- Functional coverage models are specific to a given design or family of designs
- Models cover
  - The inputs and the outputs
  - Internal states
  - Scenarios
  - Parallel properties
  - Bug Models

Functional Coverage Model Types

- Discrete set of coverage tasks
  - Set of unrelated or loosely related coverage tasks
  - In many cases, natural extension of assertions into coverage
  - Often used for corner cases
- Structured coverage models
  - The coverage tasks are defined in a structure that defines relations between the coverage tasks
    - Allow definition of similarity and distance between tasks
    - Most commonly used model types
  - Cross-product
  - Trees
  - Hybrid structures

Cross-Product Coverage Model

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

[O Lachish, E Marcus, S Ur and A Ziv. Hole Analysis for Functional Coverage Data. In proceedings of the 2002 Design Automation Conference (DAC), June 10-14, 2002, New Orleans, Louisiana, USA.]

A cross-product coverage model is composed of the following parts:

1. A semantic description of the model (story)
2. A list of the attributes mentioned in the story
3. A set of all the possible values for each attribute (the attribute value domains)
4. A list of restrictions on the legal combinations in the cross-product of attribute values

Example: Cross-Product Coverage Model 1

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Design: switch/cache unit

[G Nativ, S Mittermaier, S Ur and A Ziv. Cost Evaluation of Coverage Directed Test Generation for the IBM Mainframe. In Proceedings of the 2001 International Test Conference, pages 793-802, October 2001.]

Motivation: Interactions of core processor unit command-response sequences can create complex and potentially unexpected conditions causing contention within the pipes in the switch/cache unit when many core processors are active.

All conditions must be tested to gain confidence in design correctness.

Attributes relevant to command-response events:

– Commands - CPs to switch/cache [31]
– Responses - switch/cache to CPs [16]
– Pipes in each switch/cache [2]
– CPs in the system [8]
– (Command generators per CP chip [2])

How big is the coverage space, i.e. how many coverage tasks?

Example: Cross-Product Coverage Model 2

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Size of coverage space:

– Coverage space is formed by cross-product over all attribute value domains.

– Size of cross-product is product of domain sizes:

- 31x16x2x8x2 = 15872

– Hence, there are 15872 coverage tasks.

Example coverage task:

(Command=20, Response=01, Pipe=1, CP=5, CG=0)

Are all of these tasks reachable/legal?

– Restrictions on the coverage model are:

- possible responses for each command
- unimplemented command/response combinations
- some commands are only executed in pipe 1

– After applying restrictions, there are 1968 legal coverage tasks left.

– Make sure you identify & apply restrictions before you start!

Defining the Legal and Interesting Spaces

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

In Practice:

- The design and verification team create initial spaces based on their understanding of the design
  - Boundaries between legal and illegal are often not well understood
- Coverage feedback modifies the space definition
- Sub-models are used to economically check and refine the spaces
  - Easy to define as these are sub-crosses!
- Interesting spaces tend to change often due to shift in focus in the verification process

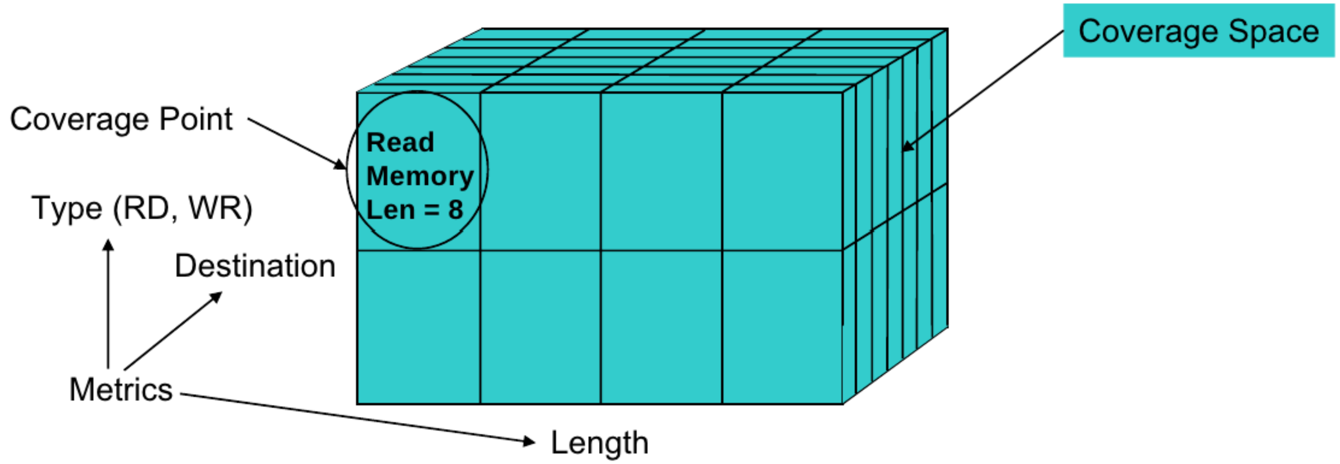
Coverage Terminology

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- coverage model: *A set of legal and interesting coverage points in the coverage space.*
- coverage point:

  1. A point within a multi-dimensional coverage space.

  2. An event of interest that can be observed during simulation.



Summary: Functional Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Determines whether the functionality of the DUV was verified.

- Functional coverage models are user-defined.
  - (specification driven)
  - This is a skill. It needs (lots of) experience!
  - Focus on control signals. WHY?
- Strengths:
  - High expressiveness: cross-correlation and multi-cycle scenarios.
  - Objective measure of progress against verification plan.
  - Can identify coverage holes by crossing existing items.
  - Results are easy to interpret.
- Weaknesses:
  - Only as good as the coverage metrics.
  - To implement the metrics, manual effort is required.

Summary: Code Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Determines if all the implementation was verified.

– Models are implicitly defined by the source code.
  - (implementation driven)
  - statement, path, expression, toggle, etc.

– Strengths:
  - Reveals unexercised parts of design.
  - May reveal gaps in functional verification plan.
  - No manual effort is required to implement the metrics. (Comes for free!)

– Weaknesses:
  - No cross correlations.
  - Can’t see multi-cycle/concurrent scenarios.
  - Manual effort required to interpret results.

Summary: Coverage Models

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Do we need both code and functional coverage? YES!!

Functional Coverage	Code Coverage	Interpretation
Low	Low	There is verification work to do.
Low	High	Multi-cycle scenarios, corner cases, cross-correlations still to be covered.
High	Low	Verification plan/or functional coverage metrics inadequate. Check for “dead” code.
High	High	High confidence of quality.

- Coverage models complement each other!
- No single coverage model is complete on its own.

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Coverage Closure

Risks of Using Coverage

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Low coverage goals
- Some coverage models are ill-suited to deal with common problems
  - Missing code
- Generating simple tests to cover specific uncovered tasks
  - There is merit in generating tests outside the coverage!
- Using coverage without commitment to using the results

Coverage Closure

Coverage closure is the process of:

– Finding areas of coverage not exercised by a set of tests.
  • Coverage Holes!
– Creating additional tests to increase coverage by targeting these holes.

Controllability Problems

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

If the cases to be hit contain internal states/signals of the DUV, directed tests that exercise all combinations are hard to find.

– Processor pipeline verification: Control logic, Internal FSMs

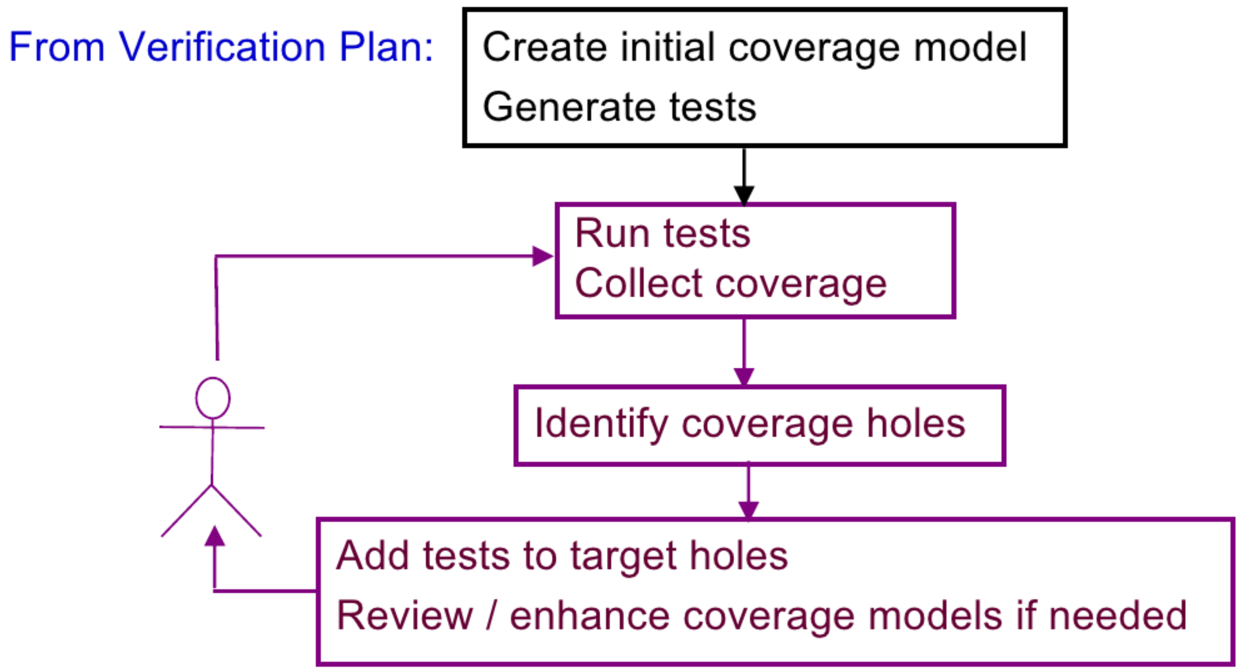
- Generate biased random tests automatically. [RTPG]
  – Typically tests are filtered to retain only those that add to coverage.
  – Coverage analysis indicates hard-to-reach cases.
  – Don't waste engineers time on what automation can achieve.
- Combine automatically generated stimulus with coverage.
- Gives rise to Coverage DRIVEN Verification Methodology

BUT:

- Hard-to-reach cases (may) need manual attention.
  – Bias tests towards certain conditions or corner cases.
  – Supplying bias requires significant engineering skill.
    - Often only trial-and-error approach.

Coverage DRIVEN Verification Methodology

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386



Current research: How can we automate this further?

80/20 Split

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

In practice: 80/20 (20/80) split wrt coverage progress.

Good news:)

– 80% of coverage is achieved (relatively quickly/easily) driving randomly generated tests.

– This takes about 20% of total time/effort/sim runs spent on verification.

Bad news:(

– Gaining the remaining 20% coverage,

- i.e. filling the remaining coverage holes (which often needs to be done manually and requires a lot of skill plus design understanding),

– can take as much as 80% of the total time/effort/sim runs spent on verification.

Benefits of Coverage DRIVEN Verification Methodology

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Shortens implementation time
  – (Initial setup time)
  – Random generation covers many “easy” cases
- Improves quality
  – Focus on goals in verification plan
  – Encourages exploration/refinement of coverage models
- Accelerates verification closure
  – Refine/tighten constraints to target coverage holes

Summary: Coverage

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Coverage is an important verification tool.
  - Code coverage: statement, path, expression
  - Structural coverage: FSM
  - Functional coverage models: story, attributes, values, restrictions
  - Assertion coverage was introduced during the lecture on Assertion-based Verification.
- Combination of coverage models required in practice.
  - Code coverage alone does not mean anything!
- Verification Methodology should be coverage driven.
- Automation: Research into coverage directed test generation
- Delays in coverage closure are the main reason why verification projects fall behind schedule!