Ontology mutation testing

February 3, 2016

Cesare Bartolini

Interdisciplinary Centre for Security, Reliability and Trust (SnT),
University of Luxembourg

Outline

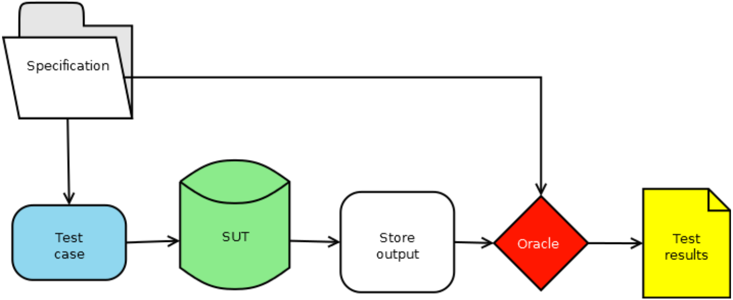
Outline

1. Mutation testing
2. Mutant generation
3. OWL ontologies
4. OWL mutation testing
5. Validation

What is testing

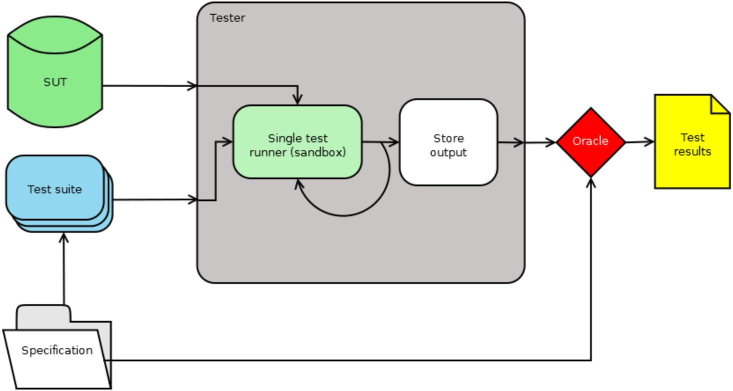
- Verifying the conformance of the System Under Test (SUT) to its requirements
- Many properties to verify
  - Correctness
  - Performance
  - Security
  - ...
- Many different ways of testing
- Requires a Test Suite (TS)
  - Manual, automated, test factory...

Testing 101



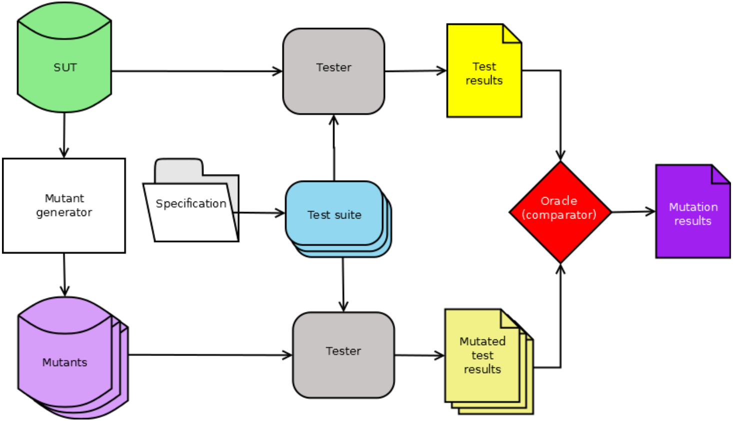
How to run a test

Testing 102



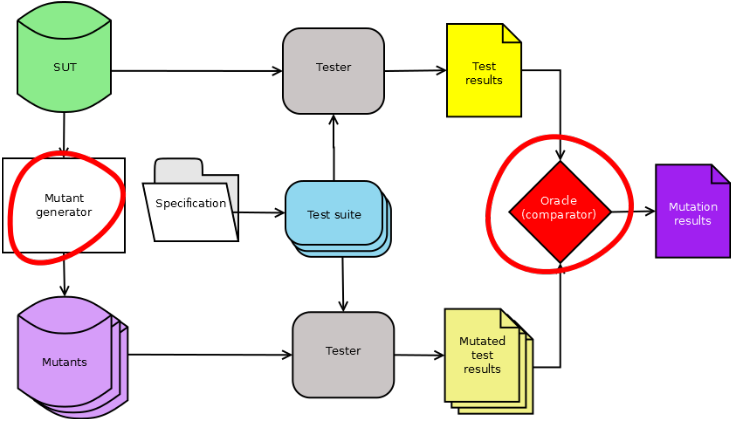
How standard testing works

Mutation testing



Mutation testing process

Mutation essentials



Mutation testing process

Step 1: normal test suite run

- Use the unmodified SUT ("golden")
- Run the test suite TS
  - Right or wrong doesn't matter!
- Store the output R_0 in some format
  - Text, XML, binary...

Step 1: normal test suite run

securityandtrust.lu

- Use the unmodified SUT ("golden")
- Run the test suite TS
  - Right or wrong doesn't matter!
- Store the output R_0 in some format
  - Text, XML, binary...

Important!

Tests should not fail (i.e., break execution) against the "golden" SUT.

Step 1: normal test suite run

- Use the unmodified SUT ("golden")
- Run the test suite TS
  - Right or wrong doesn't matter!
- Store the output R_0 in some format
  - Text, XML, binary...

Important!

Tests should not fail (i.e., break execution) against the "golden" SUT.

Consequently

It's important to fix the TS first.

Step 2: generate the mutants

- Start from ground string ("golden" SUT)
- Mutation operators
- Remove equivalent mutants (optional)
- Reduce number of mutants (optional)
- Store the mutated SUTs
- Have *n* mutants at the end

Step 3: mutant runs

- Batch runner
- Fetches a mutant
- Runs TS against the mutant
- Stores the results $R_1, \ldots, R_n$
- Rinse & repeat

Step 3: mutant runs

- Batch runner
- Fetches a mutant
- Runs TS against the mutant
- Stores the results $R_1, ..., R_n$
- Rinse & repeat

Complexity

Mutation testing can be very hard. Think of a TS with 100 tests run over a code which generates 10K mutants.

Step 4: check the outputs

- Oracle compares results
  - $R_1, ..., R_n$ against R_0
- Comparison may be difficult
- Results differ: mutant is killed
- Results do not differ: mutant is alive
- Best result: 100% killed mutants

Step 4: check the outputs

securityandtrust.lu

- Oracle compares results
  - $R_1, ..., R_n$ against R_0
- Comparison may be difficult
- Results differ: mutant is killed
- Results do not differ: mutant is alive
- Best result: 100% killed mutants

Important!

Tests may fail (i.e., execution breaks) against the mutant. The result is different from the "golden" anyway.

- E.g., if the mutant introduces an infinite loop

Step 5: and now?

SnT
securityandtrust.lu

- Mutation testing tells me how good my test suite is
  - Find patterns of live mutants
- But it can also give me insights on the SUT
- Example: mutants alive because path not covered
  - Reason 1: missing a test in the TS (must add tests)
  - Reason 2: unreachable code (must modify the SUT)
- Analysis can be complex
- Generally used for unit testing

Outline

securityandtrust.lu

1. Mutation testing
2. Mutant generation
3. OWL ontologies
4. OWL mutation testing
5. Validation

How mutant generation works

- Based on error testing or fault testing
- Hypothesis: the original SUT is correct
- Inject an error in the code
  - A single error
  - E.g., remove a semicolon
- Each error injection is a separate mutant
- Alive mutant $\Leftrightarrow$ TS cannot detect the error
  - Specific tests should be added

Semantic mutant generation

securityandtrust.lu

- Traditional mutant generation is syntactic
- Can operate on the semantics
  - E.g., + changed to -
- The system is still formally correct
- But now it should behave differently from the "golden"
- If it doesn't, then
  - TS doesn't even go there, or
  - TS goes there but code is irrelevant

This can be an error in the code or in the test suite!

Typical mutation operations

securityandtrust.lu

- Remove statement
- Change variable type
- Change unary operators
- Change arithmetical operators
- Change comparison operators
- Change logical operators
- Reverse conditions
- Reverse *then* and *else* branches
- Change 1 into 0
- ...

Typical mutation operations

securityandtrust.lu

- Remove statement
- Change variable type
- Change unary operators
- Change arithmetical operators
- Change comparison operators
- Change logical operators
- Reverse conditions
- Reverse *then* and *else* branches
- Change 1 into 0
- ...

Important!

Never use random changes.

Equivalent mutants

securityandtrust.lu

- Mutants are supposed to be different
- Two different mutants might behave identically

Example

```
for (int i = 1; i < n; i++) // "golden"
for (int i = 0; i < n; i++) // Mutant 1
for (int i = 1; i <= n; i++) // Mutant 2
```

Equivalent mutants

securityandtrust.lu

- Mutants are supposed to be different
- Two different mutants might behave identically

Example

```
for (int i = 1; i < n; i++) // "golden"
for (int i = 0; i < n; i++) // Mutant 1
for (int i = 1; i <= n; i++) // Mutant 2
```

Techniques allow equivalent detection.

Too many mutants?

- If TS is changed, mutation testing should be redone
- Possibly too much computation
- It may be necessary to further reduce the number of mutants
- Heuristics or algorithms such as Category Partition

Outline

securityandtrust.lu

What is the Web Ontology Language (OWL)?

OWL essentials

- Knowledge representation
- *Ontologies* are descriptions of a knowledge domain
- RDF is too low-level
- OWL derives from DAML+OIL
- Representation of real-world objects
- Ontologies do not *define* anything
  - Objects are defined in the domain itself
- Ontologies *describe* relations
  - By means of axioms

OWL classification

Syntax

Abstract modelling with no mandatory syntax. Possibilities:

- RDF/XML (standard, XML-based, W3C)
- OWL/XML (uses own tagset, XML-based, W3C)
- Manchester (highly descriptive, almost textual)
- Turtle (descriptive, similar to SPARQL syntax)
- ...

OWL classification

Syntax

Abstract modelling with no mandatory syntax. Possibilities:

- RDF/XML (standard, XML-based, W3C)
- OWL/XML (uses own tagset, XML-based, W3C)
- Manchester (highly descriptive, almost textual)
- Turtle (descriptive, similar to SPARQL syntax)
- ...

Semantics (OWL 2)

- OWL Full
- OWL-DL
- Several *profiles*

OWL classification

Syntax

Abstract modelling with no mandatory syntax. Possibilities:

- RDF/XML (standard, XML-based, W3C)
- OWL/XML (uses own tagset, XML-based, W3C)
- Manchester (highly descriptive, almost textual)
- Turtle (descriptive, similar to SPARQL syntax)
- ...

Semantics (OWL 2)

- OWL Full
- OWL-DL
- Several *profiles*

Syntax and semantics are irrelevant for the present work.

OWL structure

securityandtrust.lu

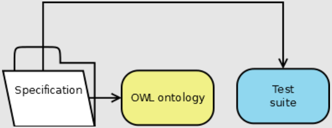
- Entities (named or anonymous)
  - Classes
  - Individuals
  - Object properties
  - Data properties
  - Datatypes
  - Annotations
  - ...
- Axioms
  - Subclass
  - Domain
  - Range
  - Class assertion
  - ...

Outline

OWL mutation

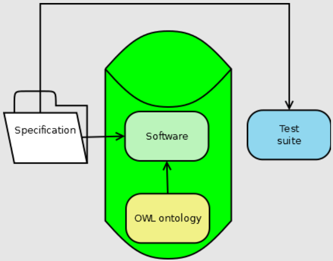
Perspectives

OWL mutation testing basics



- The SUT is the ontology
- TS built for the ontology
- E.g., SPARQL queries
- The tester must be able to run tests for the specific SUT
- E.g., SPARQL engine

A more practical perspective



- The SUT is the software
- TS built for the software
- E.g., input values for the program
- The tester only needs to run the software
- E.g., batch execution

Differences

Testing the ontology

- Deeper analysis of the ontology
- Harder to develop tests (no specific functionality)
- Harder to say when the output is wrong
- Harder to compare results (ask later)
- The testing setup is more complex because OWL does not execute

Differences

SnT
securityandtrust.lu

Testing the ontology

- Deeper analysis of the ontology
- Harder to develop tests (no specific functionality)
- Harder to say when the output is wrong
- Harder to compare results (ask later)
- The testing setup is more complex because OWL does not execute

Testing the software

- Focus only on the software requirements
- Plenty of test generation methodologies
- Outputs are clearer
- Easy to compare outputs (the software has an output format)
- The testing setup must only invoke the program

Mutation operators: categories

- Five categories of operators
  - Entities in general (E)
  - Classes (C)
  - Object properties (O)
  - Data properties (D)
  - Named individuals (I)

Mutation operators

SnT
securityandtrust.lu

Any entity	ERE	Remove the entity and all its axioms
	ERL	Remove entity labels
	ECL	Change label language
Class	CRS	Remove a single subclass axiom
	CSC	Swap the class with its superclass
	CRD	Remove disjoint class
	CRE	Remove equivalent class
Object property	OND	Remove a property domain
	ONR	Remove a property range
	ODR	Change property domain to range
	ORD	Change property range to domain
	ODP	Assign domain to superclass
	ODC	Assign domain to subclass
	ORP	Assign range to superclass
	ORC	Assign range to subclass
	ORI	Remove inverse property
Data property	DAP	Assign property to superclass
	DAC	Assign property to subclass
	DRT	Remove data type
Individual	IAP	Assign to superclasses
	IAC	Assign to subclasses
	IRT	Remove data type

Some examples

ERE operator

- Completely removes an entity
- Also removes all axioms associated with it
- If it's a class, its subclasses become subclasses of *Thing*

Some examples

securityandtrust.lu

ERE operator

- Completely removes an entity
- Also removes all axioms associated with it
- If it's a class, its subclasses become subclasses of *Thing*

OND operator

- Removes a domain from an object property
- The object property actually expands its domain

Outline

securityandtrust.lu

Experimental setup

- Programming language: Java 7+
  - Just because I haven't learnt lambda expressions yet
- Mutant generator: based on OWL API 4
- SUT is the OWL ontology in RDF/XML format
- TS is set of SPARQL queries
- Query engine: based on Apache Jena/ARQ
- `https://github.com/guerret/lu.uni.owl.mutatingowls`

Experimental setup

- Programming language: Java 7+
  - Just because I haven't learnt lambda expressions yet
- Mutant generator: based on OWL API 4
- SUT is the OWL ontology in RDF/XML format
- TS is set of SPARQL queries
- Query engine: based on Apache Jena/ARQ
- `https://github.com/guerret/lu.uni.owl.mutatingowls`

Why two libraries?

I had already developed a tool for operating on ontologies using OWL API, but OWL API does not manage SPARQL.

How the testing works

1. Generate the mutants
   1.1 Why this step first?
2. Run all queries on "golden" ontology
3. Store the results (not as text)
4. For each mutant:
   4.1 Run all queries on the mutant
   4.2 Compare against the "golden" results
   4.3 Reset the ground results
   4.4 Store if the mutant is killed or alive
5. Output a detailed report

Result comparison

securityandtrust.lu

- Mutation testing normally compares text
- SPARQL results may have a different order of the output
- Text is not an option
- Better to compare the mutants one by one
  - Too much space needed to store all results
- Jena/ARQ has the order-neutral method
  - *ResultSetCompare.equalsByTerm*
- But I must reset the "golden" results after each comparison
- By default, parsing "consumes" the data

Result comparison

- Mutation testing normally compares text
- SPARQL results may have a different order of the output
- Text is not an option
- Better to compare the mutants one by one
  - Too much space needed to store all results
- Jena/ARQ has the order-neutral method
  - *ResultSetCompare.equalsByTerm*
- But I must reset the "golden" results after each comparison
- By default, parsing "consumes" the data

Why this?

Example

securityandtrust.lu

- Tried to reuse existing stuff, avoid bias
- Reference SUT: the pizza ontology
  - `http://protege.stanford.edu/ontologies/pizza/pizza.owl`
- Set of SPARQL queries: not immediately available
  - Found `https://code.google.com/p/twouse/wiki/SPARQLASExamples`
  - Had to convert back to SPARQL
  - Very minimal, had to introduce two additional tests

Results

Operator	Mutants killed	Total mutants	Percentage
ERE	108	112	96.43
ERL	95	95	100.00
ECL	95	95	100.00
CRS	255	255	100.00
CSC	83	83	100.00
CRD	471	753	62.55
CRE	41	41	100.00
OND	0	6	0.00
ONR	0	7	0.00
ODR	0	6	0.00
ORD	0	7	0.00
ODP	0	6	0.00
ODC	1	250	0.40
ORP	0	7	0.00
ORC	1	253	0.40
IRT	0	10	0.00
Other operators	0	0	0.00
Total	1150	1986	57.62

Some preliminary analyses

securityandtrust.lu

Considerations on the TS

- The TS mainly covers the class hierarchy
  - More tests needed for properties and individuals
- Tests cover only a branch of the class hierarchy
  - Tests needed for the rest

Considerations on the SUT

- Some object properties are not used anywhere
  - This might mean they are irrelevant

Future developments

- Full-fledged test suite
  - Using both the ontology and the software relying on it as SUT
- Extend the set of mutation operators, e.g.:
  - Change the OWL cardinality constraints
  - Operate on annotations other than labels
- Algorithms to reduce the complexity (e.g., detect equivalents)
- Add a new, "structural" level of mutation (unique to ontologies), e.g.:
  - Change a subclass axiom into an object property
  - Create named classes from unnamed ones
  - Split intersections into separate entities
  - ...

Future developments

securityandtrust.lu

- Full-fledged test suite
  - Using both the ontology and the software relying on it as SUT
- Extend the set of mutation operators, e.g.:
  - Change the OWL cardinality constraints
  - Operate on annotations other than labels
- Algorithms to reduce the complexity (e.g., detect equivalents)
- Add a new, "structural" level of mutation (unique to ontologies), e.g.:
  - Change a subclass axiom into an object property
  - Create named classes from unnamed ones
  - Split intersections into separate entities
  - ...

This work will be presented at the AMARETTO workshop, co-located with the MODELSWARD conference, on February 19.