Mutation Testing in UTP

Protocol Mutation

Bernhard K. Aichernig

Institute for Software Technology
Graz University of Technology
Graz, Austria

UNU-IIST: United Nations University
International Institute for Software Technology
Macau S.A.R. China

PSSE 2007

Outline

- Introduction
- Unifying Theories of Programming (UTP)
- Formalisation: Faults and Test Cases
- Mutation Testing for Pre-postcondition Contracts
- Mutation Testing for Programs
- Mutation Testing for Protocol Specifications

Outline of this Part

Conformance Testing

- Usually focuses on structural coverage of a model
- Here: Input-Output Labelled Transition Systems (IOLTS)
- Basic method:
  1. specification with IOLTS semantics
  2. assumption: Implementation can be modeled as (input enabled) IOLTS
  3. define conformance relation
  4. generate a sound and complete set of test cases that guarantee conformance.

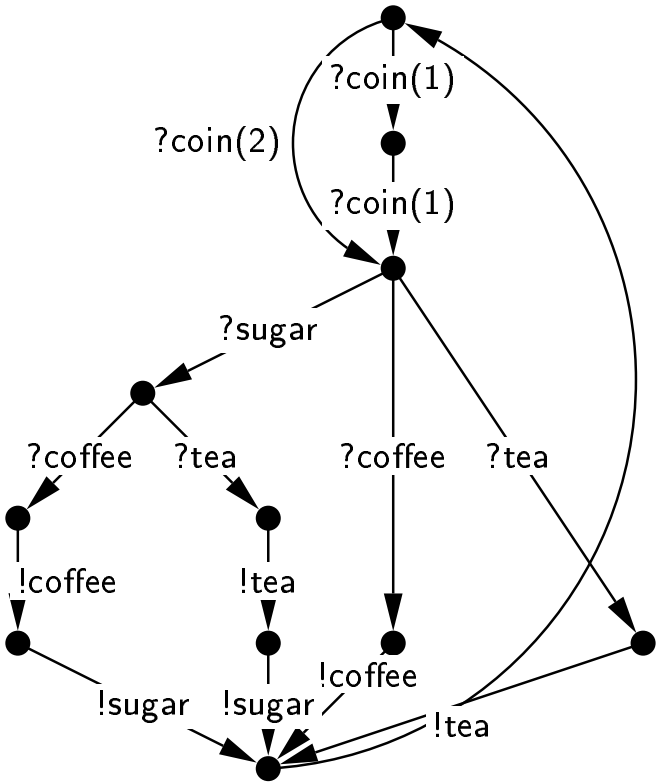
Definition (IO Conformance)

$$IUT \textbf{ ioconf } S =_{df} \forall \sigma \in Trace(S) : Out(IUT \textbf{ after}_{IUT} \sigma) \subseteq Out(S \textbf{ after}_{S} \sigma)$$

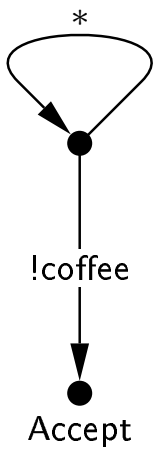
Test Purposes

- Problem: Number of test cases large!
- Existing Solution: Test purposes
  - describe test goals which parts of the behaviour should be tested
  - (here) are deterministic IOLTS which limit the exploration of the specification LTS

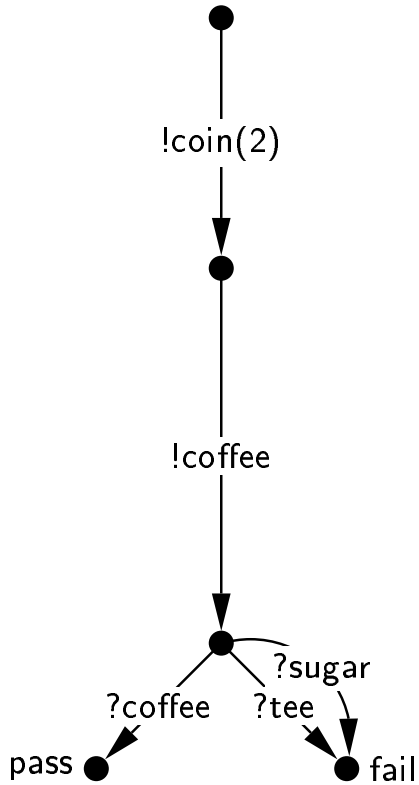
Example: Coffee Machine



model



test purpose

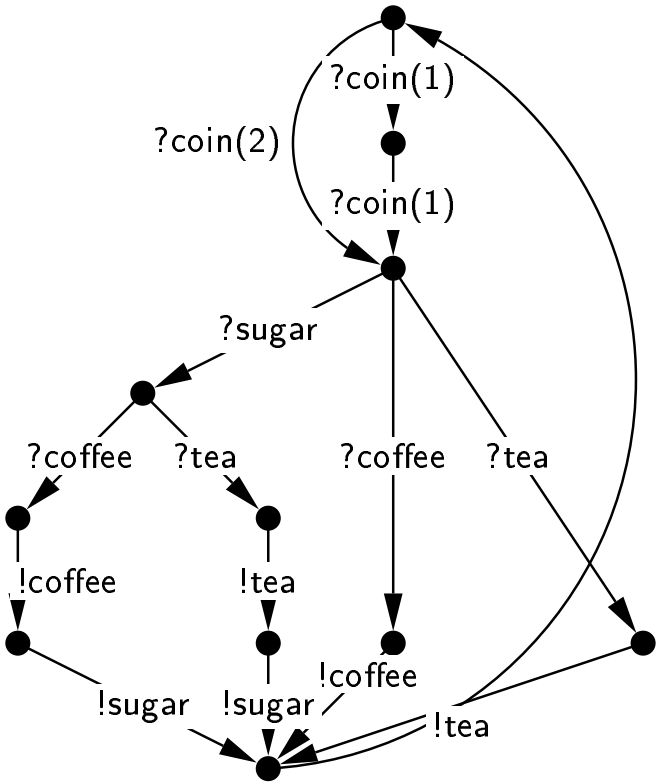


test case

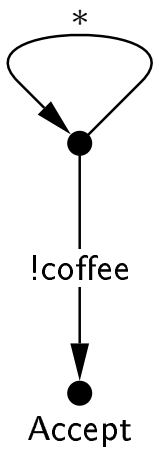
Finding Test Purposes?

- Problem: No strategy/method to find good test purposes!
- Our Solution:
  - test purposes to prevent anticipated faults
  - inject faults and generate an adequate test purpose
  - generate fault-based test cases from this test purpose

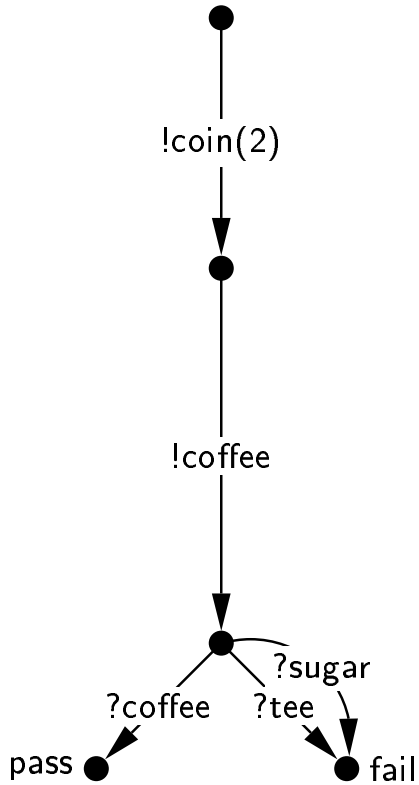
Mutating the Coffee Machine Model



model

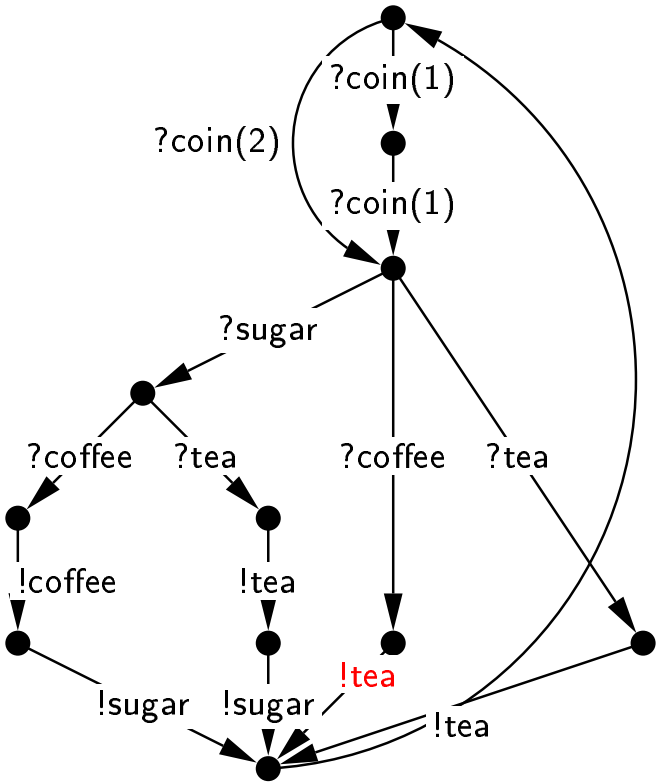


test purpose

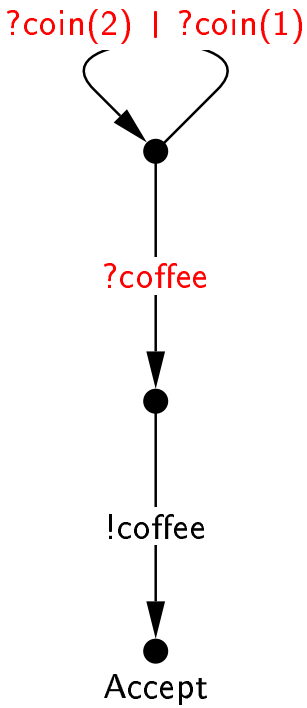


test case

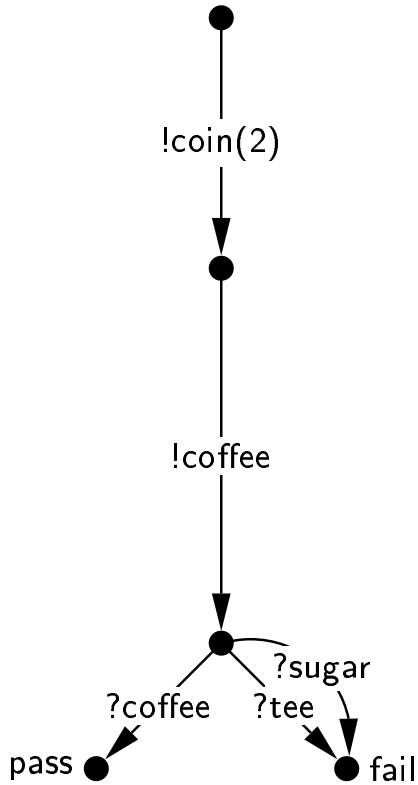
Mutating the Coffee Machine Model



model



test purpose



test case

Relating Test Purposes and Test Cases

- Test purposes are abstractions of test cases (Ledru et al.)
- Test cases (TC) are refinements of test purposes (TP)

Refinement Relation (general)

$refines(TC, TP, S)$

- in the context of a specification (S).

Relating Test Purposes and Test Cases in TGV

- TGV: test case generator of CADP toolset.
- In TGV refinement relation is defined by the properties of the test cases generated.

Refinement Relation (TGV)

$refines(generate_{TGV}(TP, S), TP, S)$

- $generate_{TGV}(TP, S)$ ... TGV test case generation algorithm

Consistency (TGV)

$consistency_{TGV}(TP, S) =_{df} refines(generate_{TGV}(TP, S), TP, S)$

Fault-based Test Purposes

- Given a specification S and a mutated (faulty) version S^m.
- Goal: Generate a test case that can distinguish S and S^m.
- Properties of such a Test Purpose:
  - Test Purpose must be consistent with S
  - Test Purpose must not be consistent with S^m

Formal Properties

$consistency_{TGV}(TP, S)$ and $\neg consistency_{TGV}(TP, S^m)$

From (Non-)Equivalence to Test Purposes

- Equivalent mutants: mutations not representing faults
- No discriminating test case $\Rightarrow$ equivalence:
  $\nexists\ TP : (consistent_{TGV}(TP, S)\ \wedge\ \neg consistent_{TGV}(TP, S^m))$
  $\Rightarrow\ (S \approx S^m)$
- which leads to our testing technique

Theorem

$(S \not\approx S^m)\ \Rightarrow$
$\exists\ TP : (consistent_{TGV}(TP, S)\ \wedge\ \neg consistent_{TGV}(TP, S^m))$

- Test purpose generation via counter-example generation of equivalence checker

Our TGV Testing Process

- Model the system under test (LOTOS). Select a mutation operator, create a mutant L^m from the original model L.
- Generate the IOLTSs S_τ and S_τ^m from the specifications L and L^m respectively (using CADP-Caesar).
- Simplify the large IOLTSs S_τ and S_τ^m to obtain S and S^M using the Safety Equivalence relation (using CADP-Aldebaran).
- Strong Bisimulation Check of the reduced IOLTS S and S^m (using CADP-Aldebaran).
- The equivalence check gives either
  - True: S^m is an equivalent mutant (no fault), no test purpose can be generated. Study the cause of equivalency.
  - False: CADP-Aldebaran issues a diagnosis (counterexample): a discriminating sequence c.

Our TGV Testing Process (cont.)

- Add the discriminating valid transition from S to the counterexample c. This sequence forms the wanted test purpose.
- Generate a test case from the discriminating test purpose (using CADP-TGV).
- Test the IUT with this test case to prevent conformance to the faulty specification L^m.
- Repeat this for every interesting mutation possible.

Mutation Operators

Symbol	Mutation Operators	Description
EDO	Event Drop Operator	Eliminate one of the events from the process definition
ESO	Event Swap Operator	Change order of the 2 neighbouring events
ERO	Event Replacement Operator	Replace event by other events
EIO	Event Insertion Operator	Inserts one event after each event in the process definition
POR	Process Operator Replacement	Replace the operator on processes (Parallel composition general case, pure interleaving and full synchronization) ||, |[]| and |||
MRO	Message Replacement Operator	Replace the message of each communication channel with other message
USO	Unobservable Sequence Operator	Change the action prefix from unobservable to observable
HDO	Hiding Delete Operator	Delete an event from hide definition
PRO	Process Replacement Operator	Replace the process name with stop or exit events
SEO	Stop and Exit interchange Operator	Interchange the Stop and Exit events
PSP	Process Swap Parameter	Change order of the two neighbouring parameters in process calls
PRP	Process Replace Parameter	Replace one parameter with other in process calls
ESP	Exit Swap Parameter	Change order of the two neighbouring parameters in Exit operator calls
...		
LRO	Logical Operators Replacement	Replace a logical operator (*and*, *or*, *not*) by another
MCO	Missing Condition Operators	Delete conditions from conjunctions, disjunctions and implications
ACO	Adding Condition Operators	Add conditions from conjunctions, disjunctions and implications
STO	Stuck-At Operators	Replace signal variables with 0 or 1

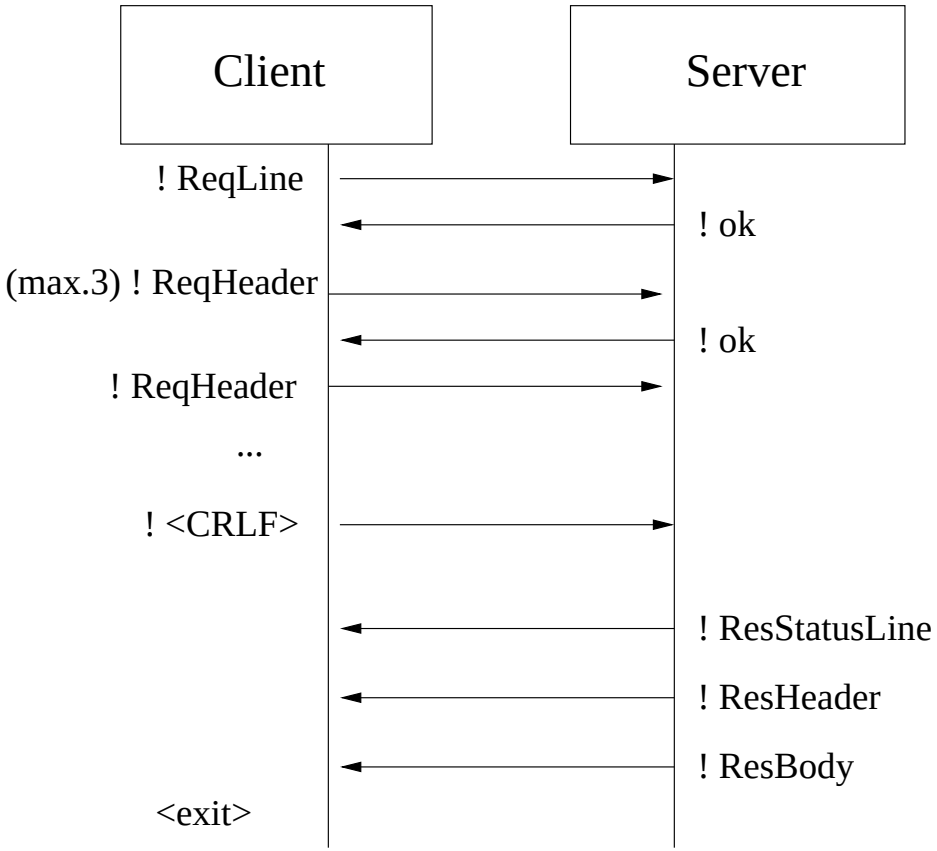
Case Study 1: Web Server Testing

- UNU-IIST (Macau)
- Bernhard K. Aichernig and Carlo Corrales Delgado: From Faults Via Test Purposes to Test Cases: On the Fault-Based Testing of Concurrent Systems. FASE, pp. 324-338, Springer, 2006.
- Tools: TGV, CADP

Case Study: HTTP Model

- Model of a subset of the http protocol (RFC 2616), client / server
- The Request message
- The Response message
- Single connection: User agent – http server.
- Subset of the protocol specifications:
  - the Method Get
  - the Headers If_Match, If_Modified_Since, If_None_Match, If_Unmodified_Since, If_Range and Range.
- Parallelism between a Server and a Client.

HTTP Client-Server Messages



Generated Mutants vs. Equivalent Mutants

Symbol	Mutant Operator	Mutants	Equiv. Mutants
EDO	Event Drop Operator	57	0
ESO	Event Swap Operator	15	0
ERO	Event Replacement Operator	65	5
EIO	Event Insertion Operator	63	7
SOR	Sequential Operator Replacement	17	7
POR	Process Operator Replacement	5	1
MRO	Message Replacement Operator	97	0
CRO	Channel Replacement Operator	46	0
USO	Unobservable Sequence Operator	2	0
HDO	Hiding Delete Operator	0	0
...			
ASO	Associative Shift Operators	48	22
	Total	1491	342

- Dead Code in the model caused many equivalent mutants.

Testing the Apache Server

- Request:
  telnet www.iist.unu.edu 80

```
GET /~carlo/index.html HTTP/1.1
Host: www.iist.unu.edu
If-Unmodified-Since: Thu, 19 Aug 2004 08:00:00 GMT
<CRLF>
```

- Response:

```
HTTP/1.1 412 Precondition Failed
Date: Fri, 27 Aug 2004 05:11:12 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Accept-Ranges: bytes
<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE html PUBLIC>
<head>  <title>Precondition failed!</title>
...
```

Result: Apache Spoke some Dialect (2006)

Example (Conditional Requests)

"An HTTP/1.1 origin server, upon receiving a conditional request that includes both a Last-Modified date (e.g., in an If-Modified-Since or If-Unmodified-Since header field) and one or more entity tags (e.g., in an If- Match, If-None-Match, or If-Range header field) as cache validators, must not return a response status of 304 (Not Modified) unless doing so is consistent with all of the conditional header fields in the request." (RFC 2616)

Result: Apache Spoke some Dialect (cont.)

- Expected Responses:

ID	*Header 1 satisfied?*	*Header 2 satisfied?*	*Response Status*
1	If-Match = true	If-Modified-Since = true	OK (200)
2	If-Match = true	If-Modified-Since = false	Not Modified (304)
3	If-Match = false	If-Modified-Since = true	Precondition Fail (412)
4	If-Match = false	If-Modified-Since = false	Precondition Fail (412)
5	If-Match = false	If-Unmodified-Since = true	Precondition Fail (412)
6	If-Match = false	If-Unmodified-Since = false	Precondition Fail (412)
7	If-None-Match = false	If-Unmodified-Since = false	Precondition Fail (412)

- Unexpected Responses:

ID	*Header 1 satisfied?*	*Header 2 satisfied?*	*Response Status*
8	If-None-Match = false	If-Modified-Since = false	Not Modified (304)
9	If-None-Match = false	If-Unmodified-Since = true	Not Modified (304)

Critical View

- Pros:
  - No additional tools needed: CADP & TGV
- Cons:
  - LTS generated from LOTOS model become extremely large.
    - In the SIP registrar case study we could not even generate the LTS.
  - Equivalence check is too strong. Wanted: Refinement!
    - Too many test cases are generated.

Case Study 2: SIP Registrar

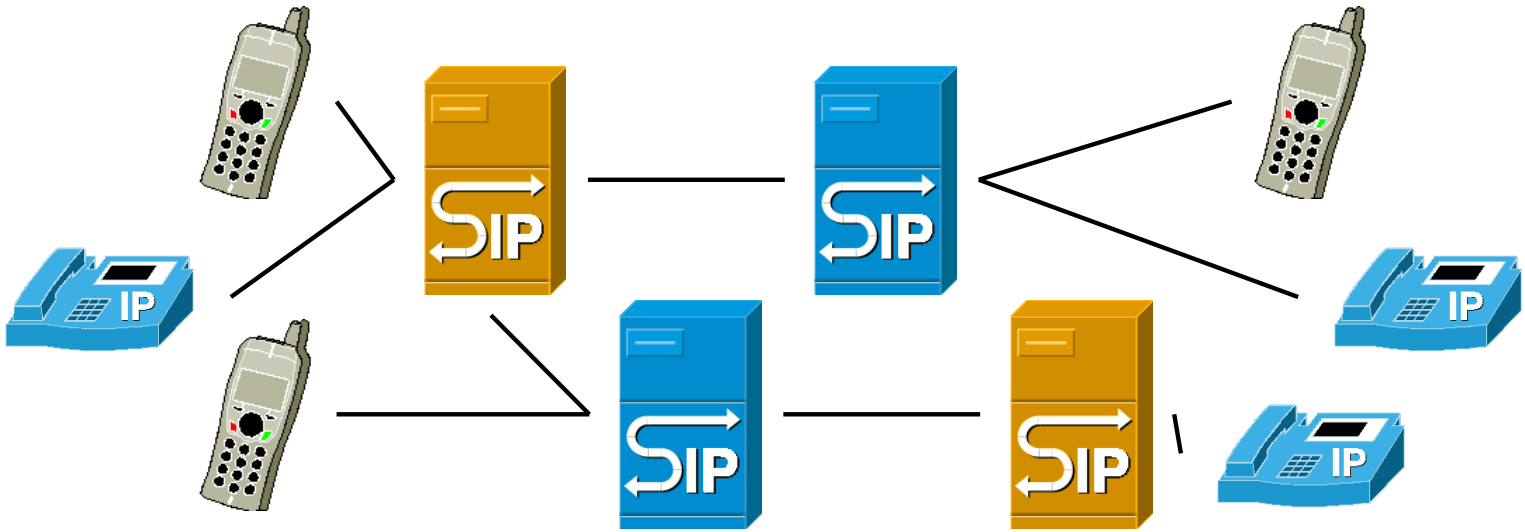
- TU Graz (Austria)
- Aichernig, B.K., Peischl, B., Weiglhofer, M., Wotawa, F.: Protocol conformance testing a SIP registrar: an industrial application of formal methods. In Hinchey, M., Margaria, T., eds.: Proceedings of the 5th IEEE International Conference on Software Engineering and Formal Methods, London, UK, IEEE (2007), 215–224.
- Tools:
  - TGV, CADP
  - mutator, ioco-checker, SIP test-driver (TU Graz)

Case Study: SIP Registrar

- Domain: Voice over IP
  - SIP: Session Initiation Protokoll
    - RFC 3261: appr. 200 pages
    - 48 additional RFC's (extensions)
    - 24 "Internet-Drafts"

Case Study: SIP Registrar

- Domain: Voice over IP
  - SIP: Session Initiation Protokoll
    - RFC 3261: appr. 200 pages
    - 48 additional RFC's (extensions)
    - 24 "Internet-Drafts"



Details of the Specification

- Session Initiation Protocol (SIP)
  - Signaling part for Voice-over-IP
  - Text-based protocol
  - User/Session Management
  - Different SIP parts (e.g., Proxy, Registrar, ...)
  - Registrar
    - Responsible for User Management
- Registrar (LOTOS) Model
  - 20 abstract data types (appr. 2.5KLOC)
  - 10 processes (appr. 500LOC)
- Testing 3 implementations
  - OpenSER - Open Source impl.
  - 2 Versions of an Industrial impl.

Results with Manually Selected Test Purposes

Test-purpose	tgv [sec]	Test cases	A		B		C	
			Pass	Fail	Pass	Fail	Pass	Fail
notfound	12	1280	16	1264	1148	132	16	1264
invalid req.	12	1328	0	1328	1008	320	0	1328
unauth.	15	880	0	880	880	0	880	0
ok	12	1488	1104	384	1104	384	1104	384
delete	7006	432	260	172	130	302	260	172
Summe	7057	5408	1380	4028	4270	1138	2260	3148

- Confirmed faults (deviations from the spec):
  - A : 9 (commercial)
  - B : 4 (OpenSER)
  - C : 8 (commercial)

Results with Mutation Testing

Op.	No. TC	A			B			C		
		pass	fail	?	pass	fail	?	pass	fail	?
EIO	22	3	18	1	0	5	17	3	8	11
EIO+	35	4	24	7	1	7	27	4	14	17
ESO	5	0	4	1	0	4	1	2	2	1
EIO/a.	22	4	0	18	6	0	16	6	0	16
EIO+/a.	35	8	0	27	8	0	27	8	0	27
ESO/a.	5	2	1	2	2	1	2	2	1	2
Total	124	21	47	56	17	17	90	25	25	74

- Additional confirmed fault (deviation from the spec):
  - A: 1, B: 0, C: 0

Results with Mutation Testing

Op.	No. TC	A			B			C		
		pass	fail	?	pass	fail	?	pass	fail	?
EIO	22	3	18	1	0	5	17	3	8	11
EIO+	35	4	24	7	1	7	27	4	14	17
ESO	5	0	4	1	0	4	1	2	2	1
EIO/a.	22	4	0	18	6	0	16	6	0	16
EIO+/a.	35	8	0	27	8	0	27	8	0	27
ESO/a.	5	2	1	2	2	1	2	2	1	2
Total	124	21	47	56	17	17	90	25	25	74

- Additional confirmed fault (deviation from the spec):
  - A: 1, B: 0, C: 0

From Faults via Non-conformance to Testcases

We are only interested in faulty mutants S^m, such that

$$\neg\ (S^m\ \textbf{ioconf}\ S)$$

Unfolding the definition of **ioconf** gives

$$= \neg\ \forall \sigma \in \mathit{Trace}(S) : \mathit{Out}(S^m\ \textbf{after}_{S^m}\ \sigma) \subseteq \mathit{Out}(S\ \textbf{after}_S\ \sigma)$$

We further simplify this criterion for failure:

$$= \exists \sigma \in \mathit{Trace}(S) : \mathit{Out}(S^m\ \textbf{after}_{S^m}\ \sigma) \not\subseteq \mathit{Out}(S\ \textbf{after}_S\ \sigma))$$

A failure is observed, if the mutant S^m produces an output o not predicted by specification S:

$$= \exists \sigma \in \mathit{Trace}(S) :\ \exists\, o :\ o \in \mathit{Out}(S^m\ \textbf{after}_{S^m}\ \sigma) \wedge o \notin \mathit{Out}(S\ \textbf{after}_S\ \sigma))$$

Consequences

- Observation 1: injecting additional inputs does not lead to failures.
- Observation 2: failure may occur, but no guarantee:

$$\exists \sigma \in Trace(S) : \exists\ o :\ o \in Out(S^m\ \mathbf{after}_{S^m}\ \sigma) \wedge o \notin Out(S\ \mathbf{after}_S\ \sigma))$$

- Observation 3: failure must occur, iff

$$\exists \sigma \in Trace(S) : \forall\ o :\ o \in Out(S^m\ \mathbf{after}_{S^m}\ \sigma) \Rightarrow o \notin Out(S\ \mathbf{after}_S\ \sigma))$$

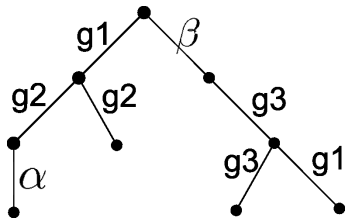
- ioco-checker
  - implemented by Martin Weiglhofer (TU Graz)
  - counter-examples serve as test purposes
  - Result: exact set of test cases

Bisimulation vs. IOCO-check

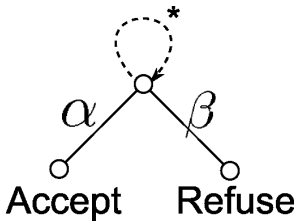
Op.	No. Marked.	Bisimulation time [s]	Bisimulation equiv.	Bisimulation diff.	tc gen.	ioco time [s]	ioco equiv.	ioco diff.	tc gen.
ASO	5	3.43	5	0	-	7.07	5	0	-
EIO	35	3.84	10	25	4.26	316.33	13	22	4.59
EIO+	35	4.48	0	35	4.75	262.43	0	35	7.83
ESO	9	4.43	4	5	3.98	8.04	4	5	3.79
MCO	11	3.91	4	7	2.68	187.5	11	0	-
Total	95	4.02	23	72	3.92	156.27	33	62	5.04

Slicing the Model with TGV

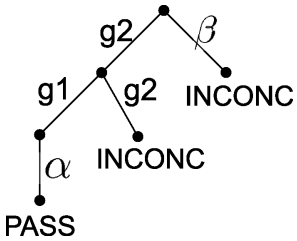
- Models too large for equivalence check, ioco check
- Observation: We know where the fault is!
- Idea: Slice away parts not relevant with TGV
  - Insert special labels and apply slicing test purpose.



(a) LTS of the marked specification.



(b) Initial slicing test purpose



(c) Resulting test graph.

- Do the equivalence or ioco check on the resulting complete test graph.

Outline

- Introduction
- Unifying Theories of Programming (UTP)
- Formalisation: Faults and Test Cases
- Mutation Testing for Pre-postcondition Contracts
- Mutation Testing for Programs
- Mutation Testing for Protocol Specifications