Using TestEra to Check the Intentional Naming System of Oxygen

Sarfraz Khurshid KHURSHID@LCS.MIT.EDU
Darko Marinov MARINOV@LCS.MIT.EDU
MIT Laboratory for Computer Science, 200 Technology Square, Cambridge MA, 02139 USA

1. Overview

We develop TestEra (Khurshid & Marinov, 2001), a novel framework for automatic generation of test data and evaluation of correctness criteria for Java programs. As an enabling technology, TestEra uses the first-order relational language Alloy and its automatic analyzer (Jackson et al., 2001). Checking a Java program using TestEra requires: 1. Creating an Alloy model of inputs to the program; 2. Creating an Alloy model expressing the correctness criteria relating inputs and output of the program; and 3. Defining concretization and abstraction translations between Alloy model valuations and Java data structures for inputs and output. TestEra gives concrete counterexamples to violated correctness criteria. Figure 1 shows the TestEra framework.

The Intentional Naming System (INS) (Adjie-Winoto et al., 1999) is the proposed naming infrastructure for dynamic resource discovery in the MIT Oxygen project. In INS, services are referred to by *intentional* names, which describe properties that services provide. An intentional name is implemented as a tree consisting of alternating levels of *attributes* and *values*, which make up the service properties. Name resolvers in INS maintain a database that stores a mapping between service descriptions and physical network locations. Client applications invoke a resolver's *Lookup-Name* method to access services of interest. Figure 2(a) illustrates an example of invoking *Lookup-Name*.

As a case study to evaluate our framework, TestEra, we analyzed the Java implementation of INS with TestEra. Here we present the flaws TestEra identified in INS. These flaws actually existed in the INS design, and we first corrected the design. Then we fixed the code and checked its correctness using TestEra. We believe TestEra presents a novel lightweight formal method for checking Java programs.

2. Checking INS using TestEra

Our checking of INS using TestEra focuses on the *Lookup-Name* method. *Lookup-Name* returns the set of services from the input database that *conform* to the input query. To investigate the correctness of *Lookup-Name*, we test its soundness (i.e., if it returns only conforming services) and completeness (i.e., if it returns all conforming services). The INS inventors did not state a formal definition of conformance, but only certain properties of *Lookup-Name*.

Due to space limitation, we do not present here the Alloy model of correctness for INS, and the concretization and abstraction translations, used in TestEra's analyses. They appear elsewhere (Khurshid & Marinov, 2001). Table 1 outlines the results of the analyses explained below.

The published description of *Lookup-Name* claims: "This algorithm uses the assumption that omitted attributes correspond to wildcards; this is true for both the queries and advertisements." TestEra refutes this claim (Figure 2(b)).

TestEra also shows that addition in INS is not monotonic, i.e., addition of a new service to a database can cause existing services to erroneously become non-conforming (Figure 2(c)). This flaw points out that INS did not have a consistent notion of conformance. Both preceding flaws exist in the original design and implementation of INS.

We define a service s as conforming to a query q if s provides all the attributes and values in q in the right order (*sub-tree*). TestEra's analysis of the original implementation of *Lookup-Name* with respect to this definition of conformance reports several counterexamples. We modified the implementation and re-evaluated the correctness of *Lookup-Name* using TestEra. This time TestEra reports no flaws, increasing the confidence that our changes have corrected the problems with INS. The corrected algorithm now forms a part of the INS code base.

References

Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., & Lilley, J. (1999). The design and implementation of an intentional naming system. *SOSP 99*. Kiawah Island.

Jackson, D., Shlyakhter, I., & Sridharan, M. (2001). A micromodularity mechanism. *FSE 01*. Vienna, Austria.

Khurshid, S., & Marinov, D. (2001). Checking Java implementation of a naming architecture using TestEra. *CAV Workshop on Software Model Checking*. Paris, France.

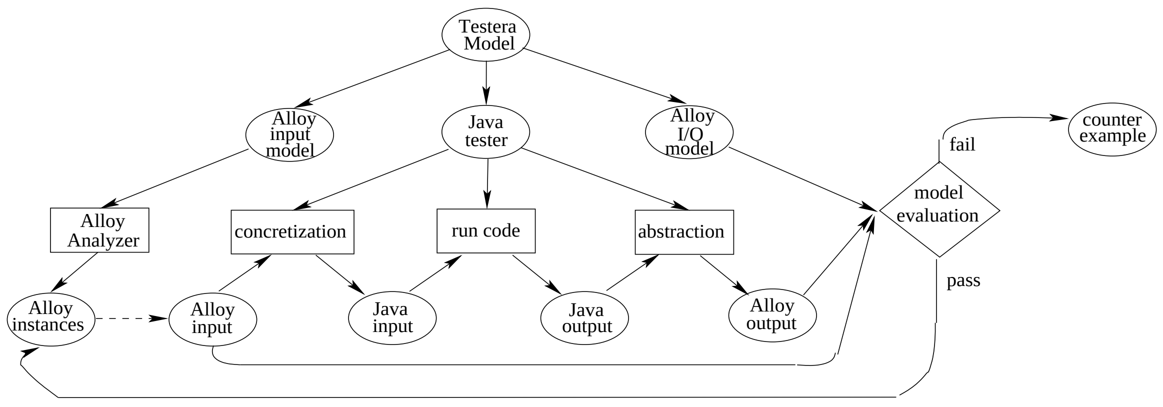

Figure 1. TestEra framework. A TestEra model consists of Alloy and Java code. TestEra's analysis proceeds in two phases: first, all non-isomorphic instances of the Alloy input model (up to a given maximum input size) are automatically generated, and second, testing is automatically performed using appropriate concretization and abstraction translations and verification of correctness properties.

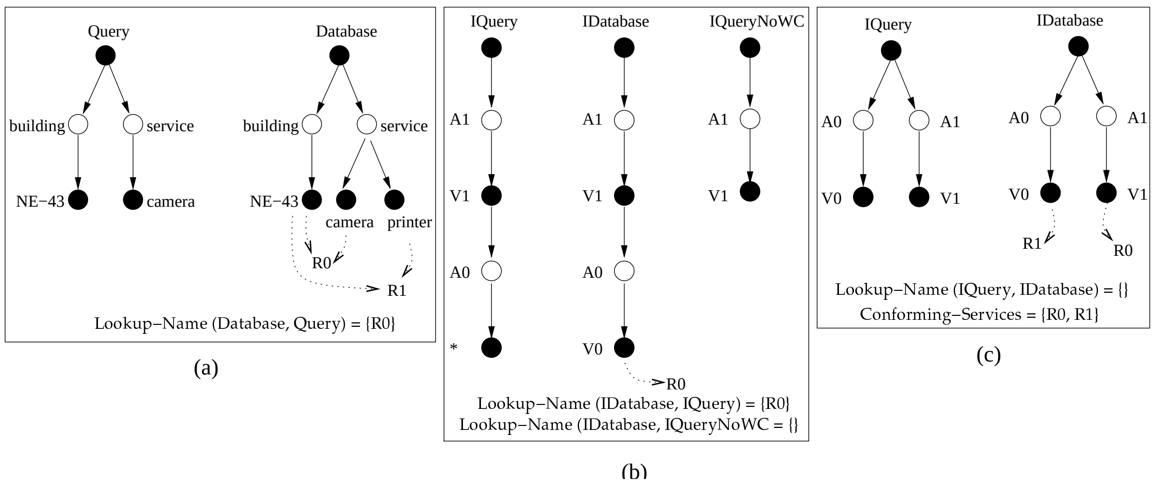
Query
Database
building
service
NE−43
camera
printer
R0
R1
Lookup−Name (Database, Query) = {R0}
(a)
IQuery
IDatabase
IQueryNoWC
A1
V1
A0
*
V0
Lookup−Name (IDatabase, IQuery) = {R0}
Lookup−Name (IDatabase, IQueryNoWC = {}
(b)
Lookup−Name (IQuery, IDatabase) = {}
Conforming−Services = {R0, R1}
(c)

Figure 2. (a) Intentional names in INS. $Query$ describes a camera service in building NE-43. $Database$ stores descriptions of two services: service $R0$ provides a camera service in NE-43 and service $R1$ provides a printer service in NE-43. Invoking *Lookup-Name* on $Query$ and $Database$ should return $R0$. (b) TestEra's counterexample to wildcard claim. $IQueryNoWC$ is the same as $IQuery$, except for the omission of the wildcarded attribute $A0$. Different results of the two invocations of *Lookup-Name* contradict the claim. (c) TestEra's counterexample to monotonicity of addition. Both services $R0$ and $R1$ are considered conforming to $IQuery$ by the semantics of INS, but their co-existence in $IDatabase$ makes both of them erroneously non-conforming to $IQuery$.

	Input size			*Phase 1*		*Phase 2*	
Property tested	*Val*	*Att*	*Ser*	*# Tests*	*t[sec]*	*# Passed*	*t[sec]*
`Published wildcard claim`	3	3	2	12	9	10 (83%)	6
`Monotonicity of addition`	4	2	2	160	14	150 (93%)	9
`Correctness (original` *`Lookup-Name`*`)`	3	3	2	16	8	10 (62%)	6
`Correctness (corrected` *`Lookup-Name`*`)`	3	3	2	16	8	16 (100%)	6

Table 1. Summary of TestEra's analyses. In all cases, TestEra takes a few seconds to complete the checking. All the properties are refuted for small input sizes. The exhaustive checking performed by TestEra uncovers subtle bugs that went undetected for over a year of use of INS.

Acknowledgements
We would like to thank Ilya Shlyakhter, Victor Kuncak, Daniel Jackson, Martin Rinard, Manu Sridharan, Radu Rugina, and Alexandru Salcianu for helpful discussions and feedback. This work was funded in part by ITR grant #0086154 from the National Science Foundation.