Model-based testing

6 December 2017

How can we obtain models for testing?

- from exploring the system
- from the specification
- from code

From models to tests

In all cases, we need a mapping from actions and responses of the model to inputs and responses of the system under test (SUT)

Example: Web Application Abstract Language [Büchler et al., KIT/TU München]
1) *Abstract* browser actions: *FollowLink*, *ClickButton*, *SelectItems*, *ClickImage*, *gotoURL*, *InputText*, *MoveMouse*, etc.

2) Mapping to actions *specific* to SUT:

```
login(user, pwd) =
  selectItem(employeeList, user);
  inputText(passwordField, pwd);
  clickButton(login);
```

3) Mapping to actions of the testing framework (e.g., Selenium):

```
  HtmlUnit.findElement(), WebElement.click()
```

Models obtained by explorinng the system

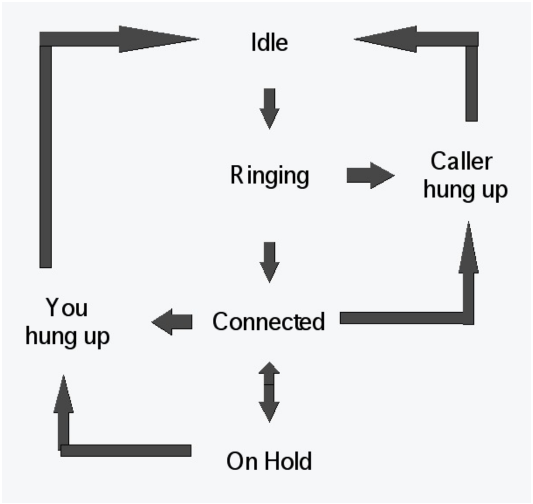
Informal: exploratory testing
  e.g., model of a GUI (file editor) and generated program actions
Model building: manually
Conformance testing (system respects model?): automated

Formal: automata learning (*active learning*, Angluin algorithm)
  generate input sequences, observing outputs
If two sequences i_1, i_2 cannot be distinguished by suffixes w up to a given length ($i_1 w$ and $i_2 w$ generate same outputs), consider they lead to the same state.

Currently very successsful in learning / testing network protocols

Models obtained from specification

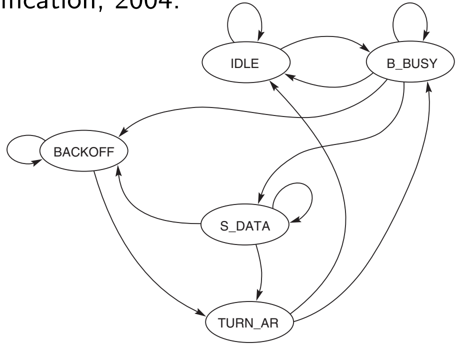
Example: phone switch [Kaner]



Usually written by hand

Models as part of specifications

PCI Local Bus Specification, 2004:



"if a conflict exists between the specification and the state machines, the specification has precedence."

IETF Extensible Authentication Protocol (EAP), FRC 4137 (2005)

"Should a conflict exist between the interpretation of a state diagram and either the corresponding global transition tables or the textual description associated with the state machine, the state diagram takes precedence. "

Models extracted from code

```
do {    // Fragment de device driver [Ball & Rajamani '01]
  KeAcquireSpinLock(&devExt->writeListLock);
  nPacketsOld = nPackets;
  request = devExt->WriteListHeadVa;
  if(request && request->status) {
    devExt->WriteListHeadVa = request->Next;
    KeReleaseSpinLock(&devExt->writeListLock);
    irp = request->irp;
    if (request->status > 0) {
      irp->IoStatus.Status = STATUS_SUCCESS;
      irp->IoStatus.Information = request->Status;
    } else {
      irp->IoStatus.Status = STATUS_UNSUCCESSFUL;
      irp->IoStatus.Information = request->Status;
    }
    SmartDevFreeBlock(request);
    IoCompleteRequest(irp, IO_NO_INCREMENT);
    nPackets++;
  }
} while (nPackets != nPacketsOld);
KeReleaseSpinLock(&devExt->writeListLock);
```

Using abstractions to obtain a model

```
do {
A: KeAcquireSpinLock();
  b = T;      /* b == (nPackets == nPacketsOld) */
  if(*) {
B:   KeReleaseSpinLock();
    if (*) {
      skip;
    } else {
      skip;
    }
    b := choose(F, b);     /* choose(p1, p2) == p1 ?  T :
p2 ?  F : nondet */
  }
} while (!b);
C: KeReleaseSpinLock();
```

Abstractions use Hoare rules / Dijkstra weakest preconditions

Abstractions from code: JML model fields

Fictitious fields, representing relations between actual object fields

Each method: annotated with preconditions / postconditions / invariants, expressed in terms of *model fields*

`http://kindsoftware.com/products/opensource/ESCJava2/ESCTools/slides/ETAPSTutorial/5_more_jml.pdf` (p. 35-45)