Testing Meets Static and Runtime Verification

Jesús Mauricio Chimento, Wolfgang Ahrendt, Gerardo Schneider

FormaliSE’18

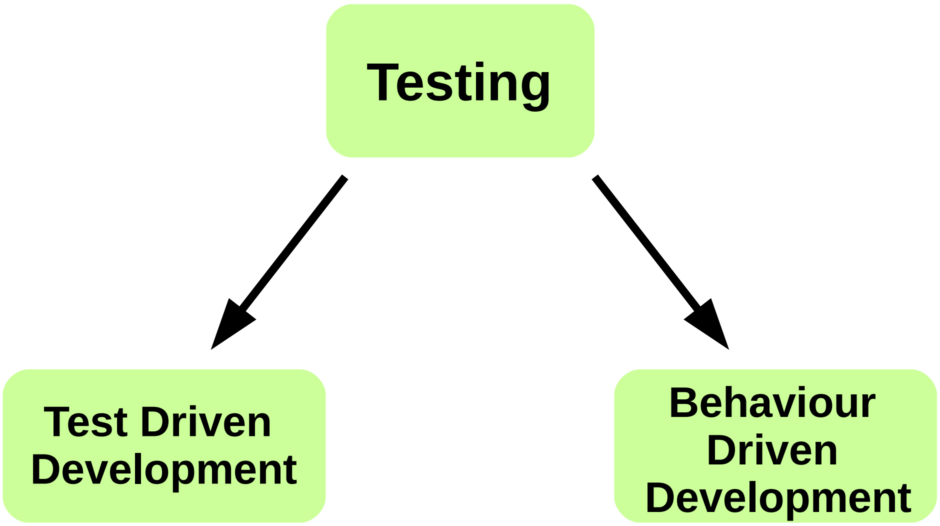
2 June 2018

Software Development

Testing

Formal Verification

Software Development



Testing

Test Driven Development

Behaviour Driven Development

Test Driven Development

Test Driven Development

- Write (unit) test cases which initially fail
- Write code making the tests pass
- *Refactor* the code

Example

- Write (unit) test cases which initially fail

```
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */
 public Object delete (int key) { }
```

Example

- Write (unit) test cases which initially fail

```
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */
 public Object delete (int key) { }
```

```
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);

    HashTable aux = new HashTable(2);
    aux.add(new Integer(3),1);

    Object res = hash.delete(0);

    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```

Example

- Write code making the tests pass

```
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */
 public Object delete (int key) {
    if (key >= 0) {
       if (h[key] == null)
          return null;
       else {
          Object ret = h[key] ;
          h[key] = null ;
          size = size - 1;
          return ret;
       }
    } else { return null; }
 }
```

```
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);

    HashTable aux = new HashTable(2);
    aux.add(new Integer(3),1);

    Object res = hash.delete(0);

    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```

Behaviour Driven Development

Behaviour Driven Development

- Red - Green - *Refactor*
- Scenarios instead of unit tests

GIVEN some condition
WHEN performing an action
THEN something should happen

Behaviour Driven Development

- Property: deposit available only when user is logged

GIVEN user is not logged
WHEN user logs successfully
THEN user is logged

GIVEN user is logged
WHEN user deposits money
THEN user is still logged

GIVEN user is logged
WHEN user logs out successfully
THEN user is not logged

```
/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */
public void deposit(int money){

}
```

Behaviour Driven Development

- Property: deposit available only when user is logged

GIVEN user is not logged
WHEN user logs successfully
THEN user is logged

GIVEN user is logged
WHEN user deposits money
THEN user is still logged

GIVEN user is logged
WHEN user logs out successfully
THEN user is not logged

```
/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */
public void deposit(int money){
     if (u != null)
         u.getAccount().deposit(money);
}
```

Behaviour Driven Development

- Property: deposit available only when user is logged

GIVEN I am on state “Logout”
WHEN I successfully log in
THEN I should be on state “Login”

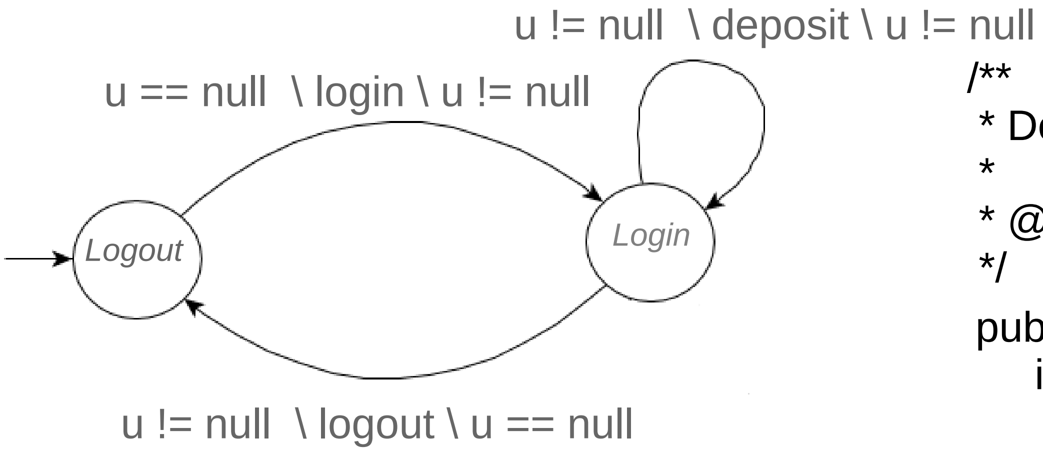
GIVEN I am on state “Login”
WHEN I deposit money
THEN I should be on state “Login”

GIVEN I am on state “Login”
WHEN I successfully log out
THEN I should be on state “Logout”

```
/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */
public void deposit(int money){
     if (u != null)
         u.getAccount().deposit(money);
}
```

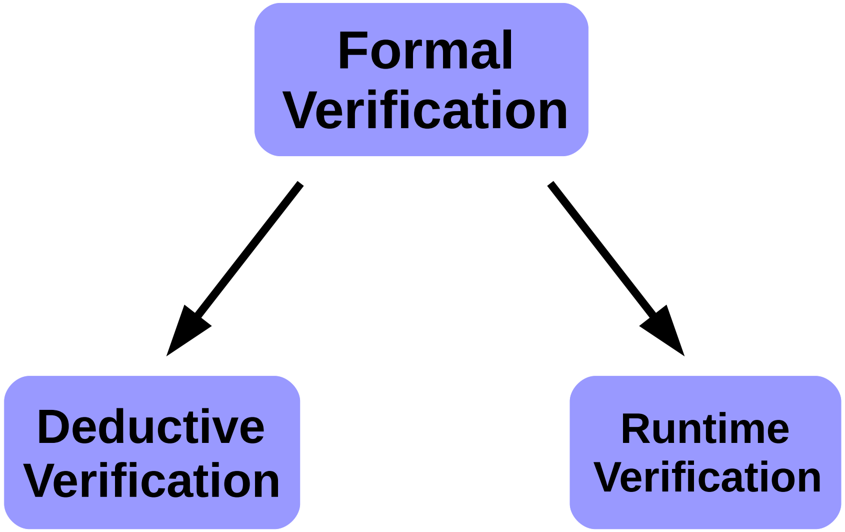
Behaviour Driven Development

- Property: deposit available only when user is logged


u != null \ deposit \ u != null
u == null \ login \ u != null
Logout
Login
u != null \ logout \ u == null

```
/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */
public void deposit(int money){
     if (u != null)
         u.getAccount().deposit(money);
}
```

Software Development



Formal Verification

Deductive Verification

Runtime Verification

Deductive Verification

- Properties written as logical formulae

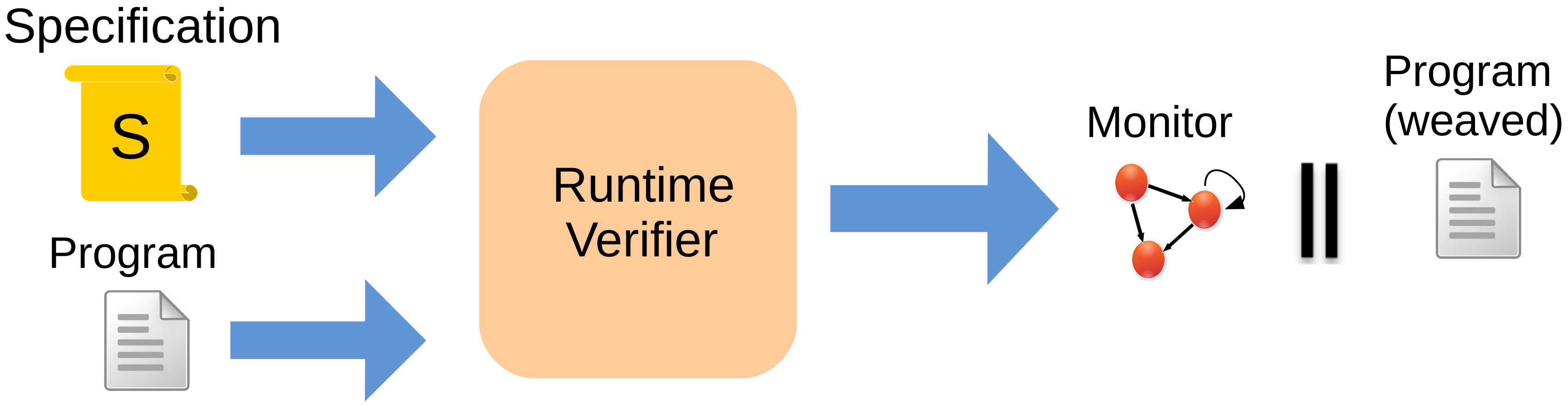
{ P } foo() { Q }

- Formulae are verified by deduction in a calculus

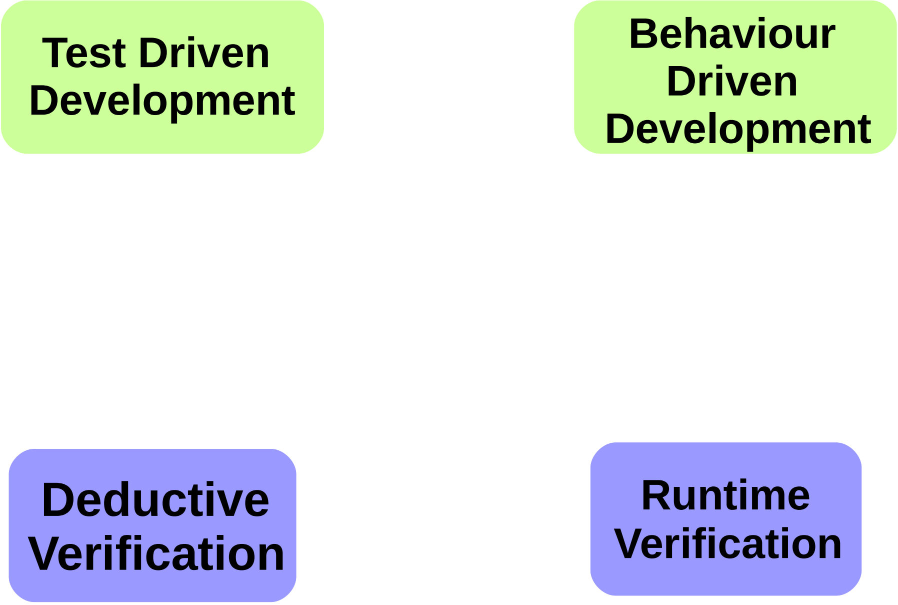
$$\frac{\Gamma, \sigma(b) \vdash \sigma < s_1 \; \omega > \phi \qquad \Gamma, \sigma(\neg b) \vdash \sigma < s_2 \; \omega > \phi}{\Gamma \vdash \sigma < \texttt{if} \; b \; s_1 \; \texttt{else} \; s_2 \; \omega > \phi}$$

Runtime Verification

- Monitoring of program executions



Test Focus Development



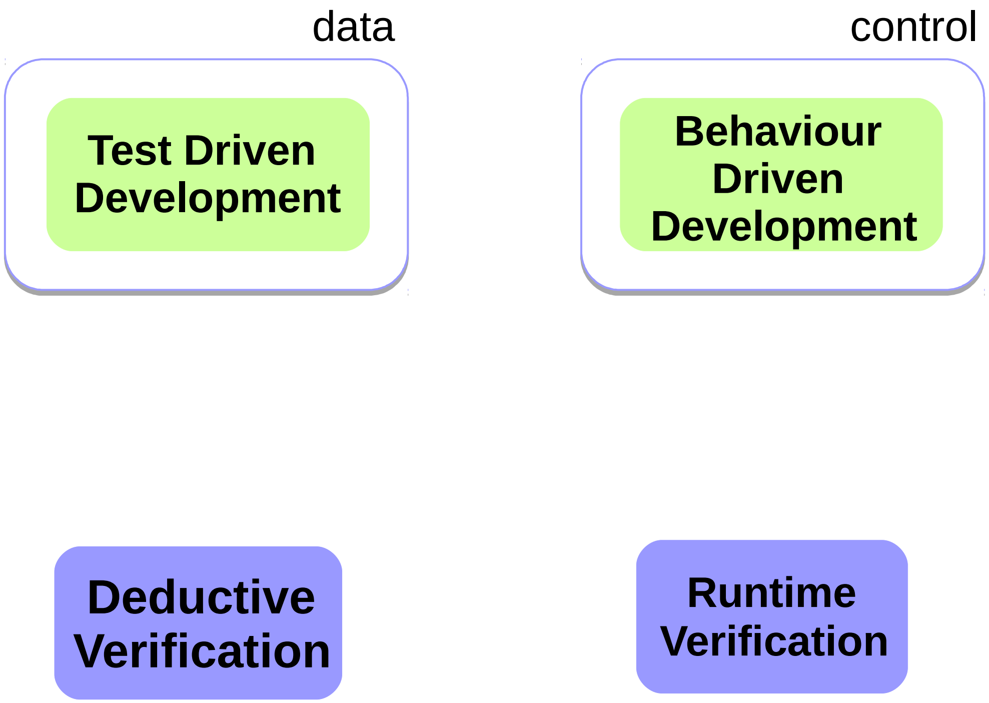
Test Driven Development

Behaviour Driven Development

Deductive Verification

Runtime Verification

Test Focus Development



data

Test Driven Development

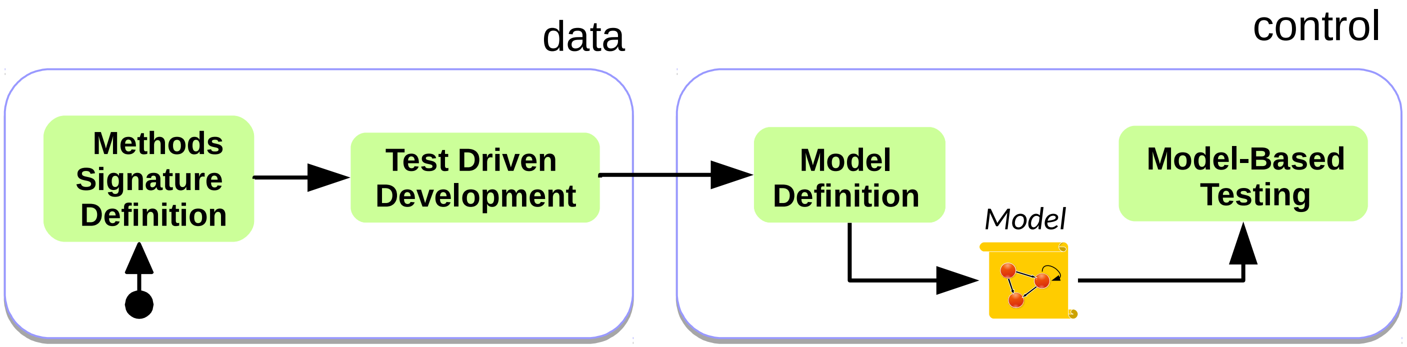
control

Behaviour Driven Development

Deductive Verification

Runtime Verification

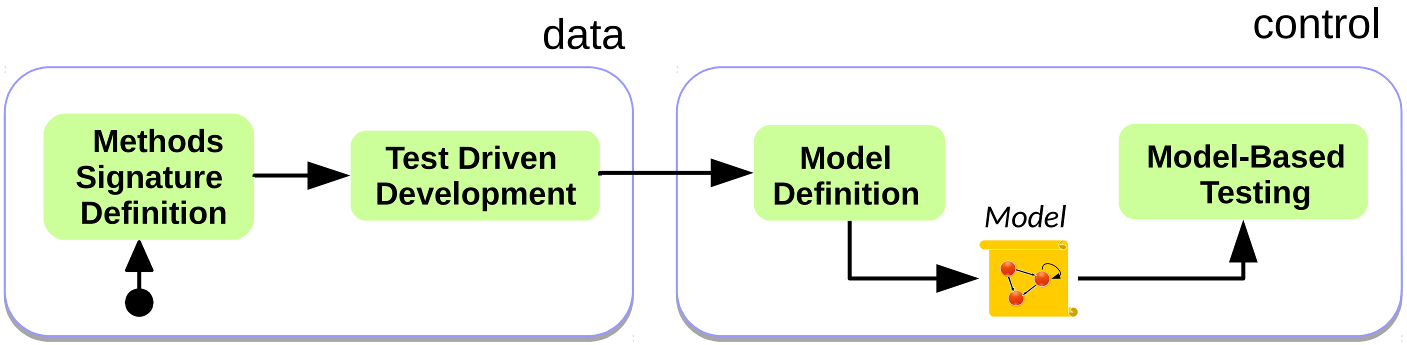
Test Focus Development



Deductive Verification

Runtime Verification

Test Focus Development



data

Methods Signature Definition

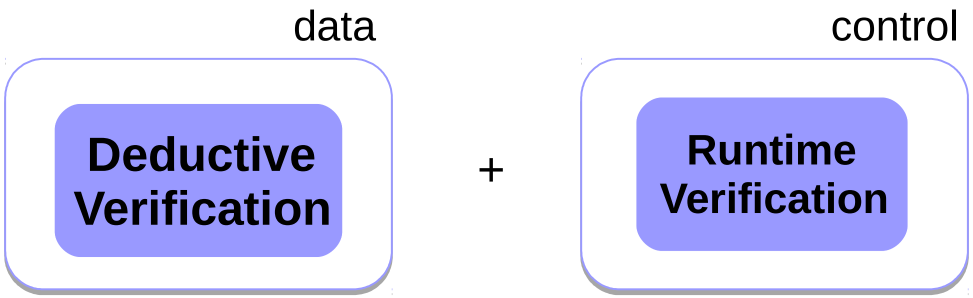
Test Driven Development

control

Model Definition

Model

Model-Based Testing



data

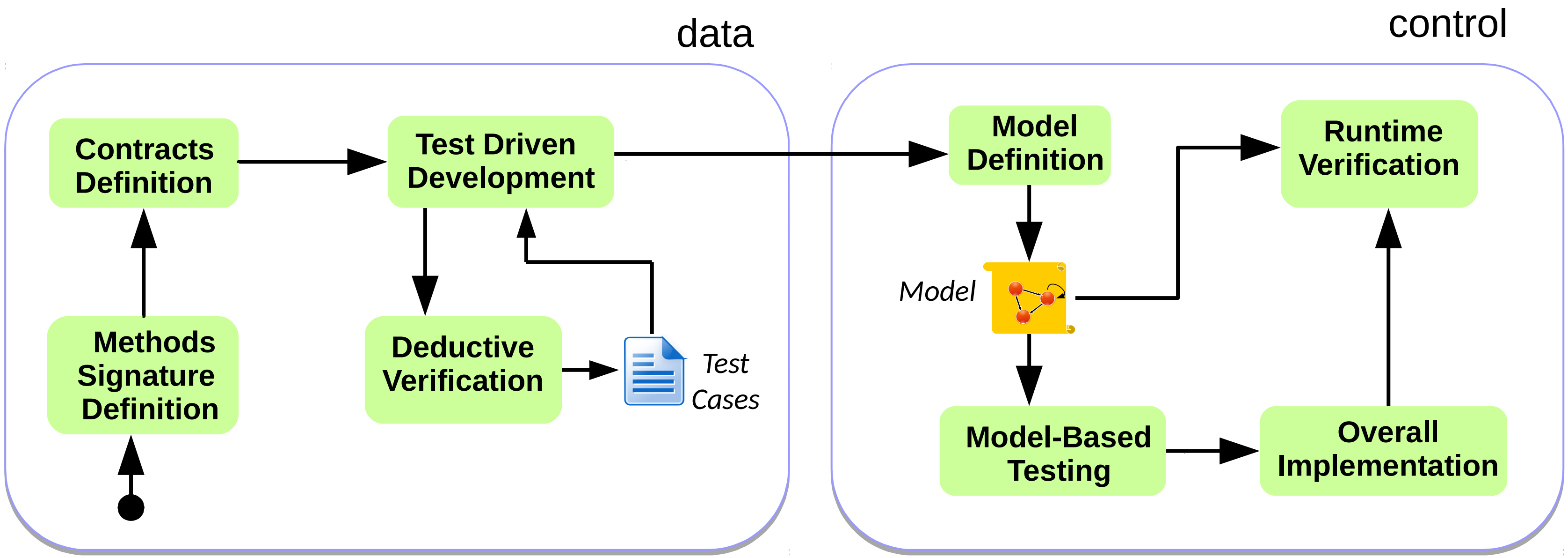
Deductive Verification

+

control

Runtime Verification

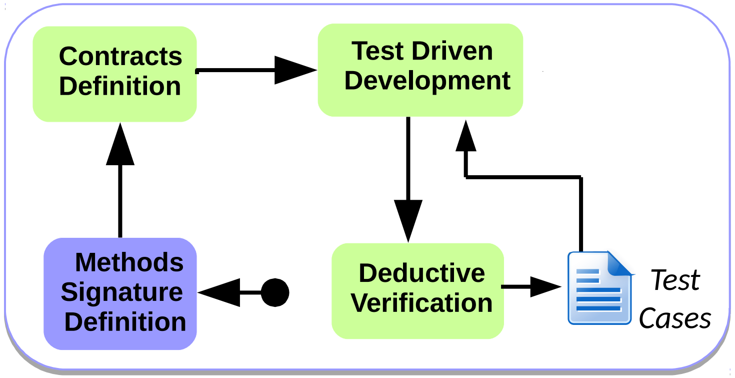
Testing Meets
Deductive and Runtime Verification



Example

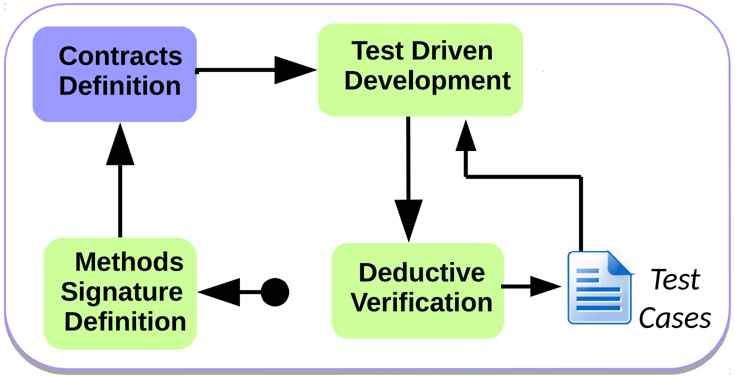
- Define the methods signature

```
/**
 * Deletes entry at <tt>key</tt> from the hashtable.
 *
 * @param key of the removed object
 * @return removed object
 */
 public Object delete (int key) { }
```



Example

- Define contracts for the methods

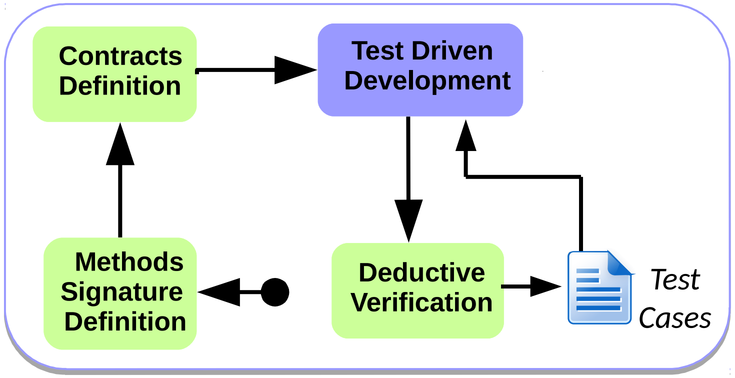


```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) { }
```

Example

- Apply TDD (at least one test per contract)

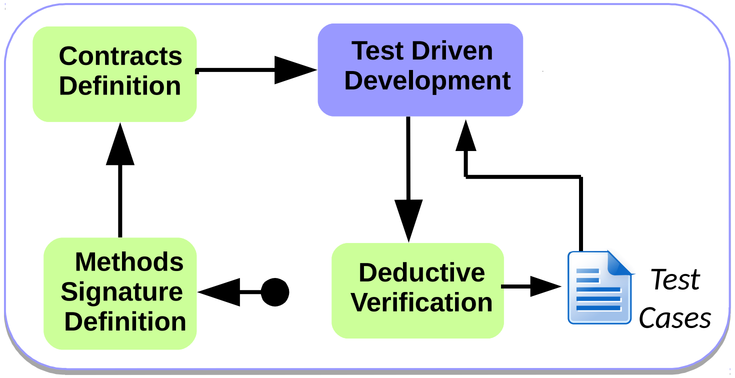
```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) { }
```



```
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);

    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3),1);

    Object res = hash.delete(0);

    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```

Example

- Apply TDD (at least one test per contract)



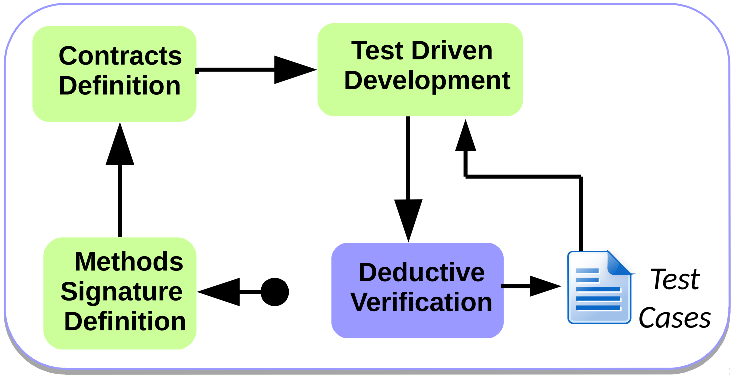
```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) {
    if (key >= 0) {
      if (h[key] == null)
        return null;
      else {
        Object ret = h[key];
        h[key] = null;
        size = size - 1;
        return ret;
      }
    } else { return null; }
  }
```

```
@Test
public void test_delete_1(){
    hash.add(new Integer(42),0);
    hash.add(new Integer(3),1);

    HashTable aux = new HashTable(3) ;
    aux.add(new Integer(3),1);

    Object res = hash.delete(0);

    assertEquals(res,new Integer(42));
    assertNull(hash.get(0));
    assertTrue(hash.size == 1);
    assertArrayEquals(aux.h, hash.h);
}
```

Example

- Deductive verify the implementation

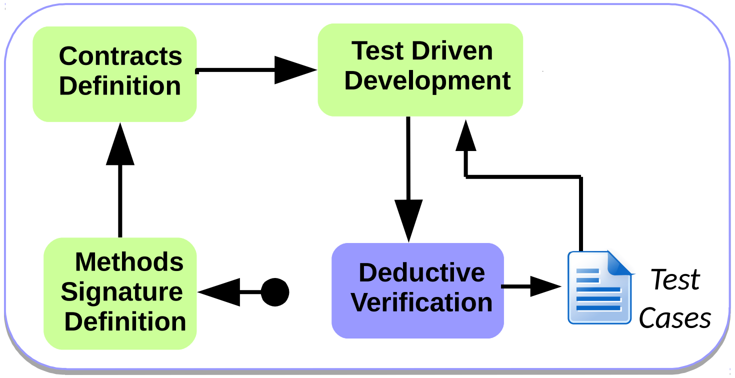
```
/*@ public normal_behaviour
 @ requires key >= 0 ;
 @ requires h[hash(key)] != null ;
 @ requires size > 0 ;
 @ ensures \result == \old(h[hash(key)]) ;
 @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
 @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
 @ assignable size,h[*] ;
 @ also
 .....
 @*/
 public Object delete (int key) {
   if (key >= 0) {
     if (h[key] == null)
       return null;
     else {
       Object ret = h[key];
       h[key] = null;
       size = size - 1;
       return ret;
     }
   } else { return null; }
 }
```



Example

- Proof-based test case generation

```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) {
    if (key >= 0) {
      if (h[key] == null)
        return null;
      else {
        Object ret = h[key];
        h[key] = null;
        size = size - 1;
        return ret;
      }
    } else { return null; }
  }
```

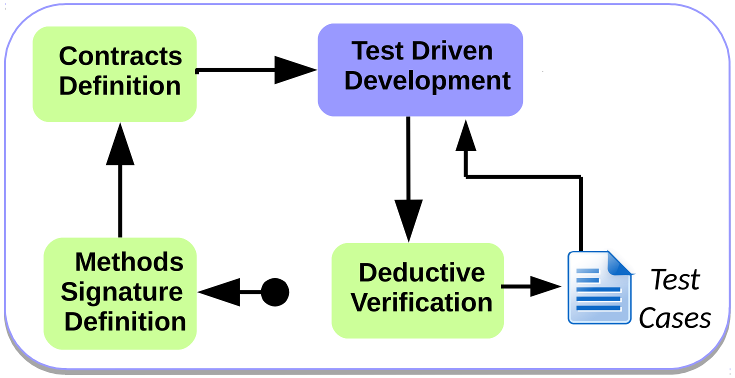


KeyTestGen

Example

- Proof-based test case generation

```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) {
    if (key >= 0) {
      if (h[key] == null)
        return null;
      else {
        Object ret = h[key];
        h[key] = null;
        size = size - 1;
        return ret;
      }
    } else { return null; }
  }
```

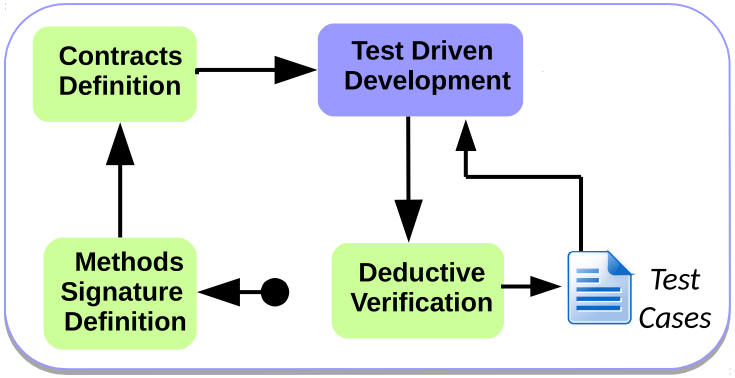


KeyTestGen generates a (failing) test case which throws an index out of bound exception.

Example

- Proof-based test case generation

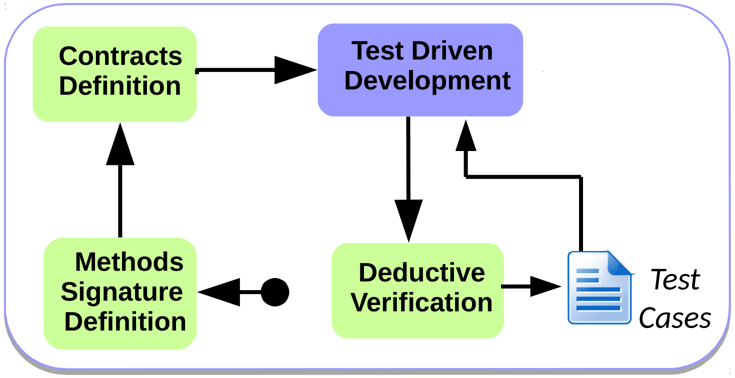
```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) {
     if (key >= 0) {
        int i = hash(key);
        if (h[i] == null)
           return null;
        else {
           Object ret = h[i];
           h[i] = null;
           size = size - 1;
           return ret;
        }
     } else { return null; }
  }
```



KeyTestGen generates a (failing) test case which throws an index out of bound exception.

Example

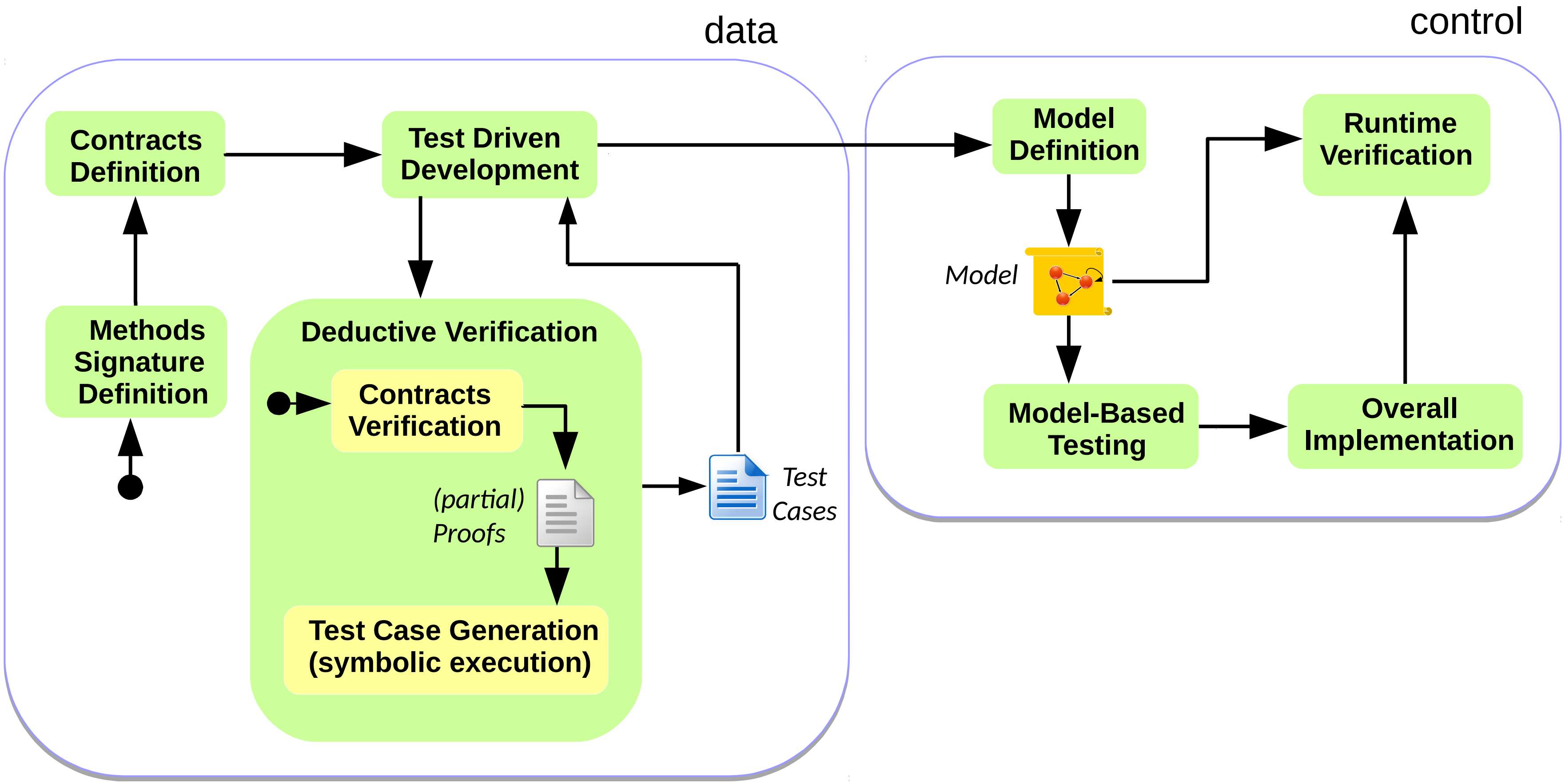
- Proof-based test case generation



```
/*@ public normal_behaviour
  @ requires key >= 0 ;
  @ requires h[hash(key)] != null ;
  @ requires size > 0 ;
  @ ensures \result == \old(h[hash(key)]) ;
  @ ensures h[hash(key)] == null && size == \old(size) – 1 ;
  @ ensures (\forall int j; j >= 0 && j < capacity && j != hash(key) ; h[j] == \old(h[j])) ;
  @ assignable size,h[*] ;
  @ also
  …..
  @*/
  public Object delete (int key) {
     if (key >= 0) {
        int i = hash(key);
        if (h[i] == null)
           return null;
        else {
           Object ret = h[i];
           h[i] = null;
           size = size - 1;
           return ret;
        }
     } else { return null; }
  }
```

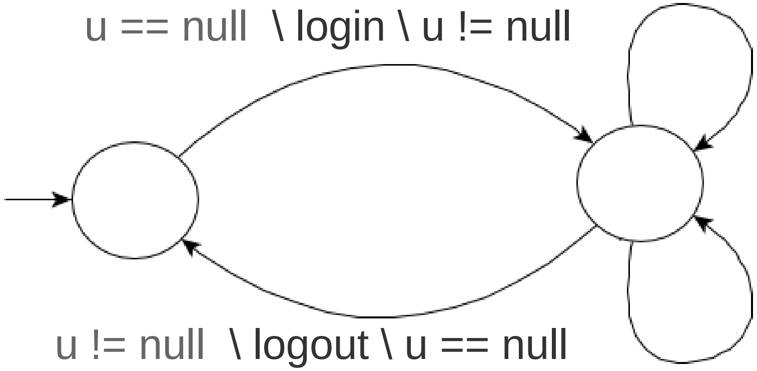
```
@Test
public void test_delete_1(){
   hash.add(new Integer(42),0);
   hash.add(new Integer(3),1);

   HashTable aux = new HashTable(3) ;
   aux.add(new Integer(3),1);

   Object res = hash.delete(0);

   assertEquals(res,new Integer(42));
   assertNull(hash.get(0));
   assertTrue(hash.size == 1);
   assertArrayEquals(aux.h, hash.h);
}
```

Testing Meets Deductive and Runtime Verification



Example

- Define the model for your (control) property

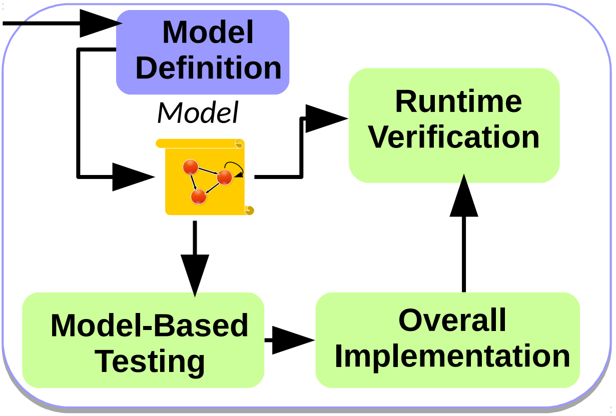


u != null \ deposit \ u != null

u == null \ login \ u != null

u != null \ logout \ u == null

u != null \ withdraw \ u != null



Model Definition

Model

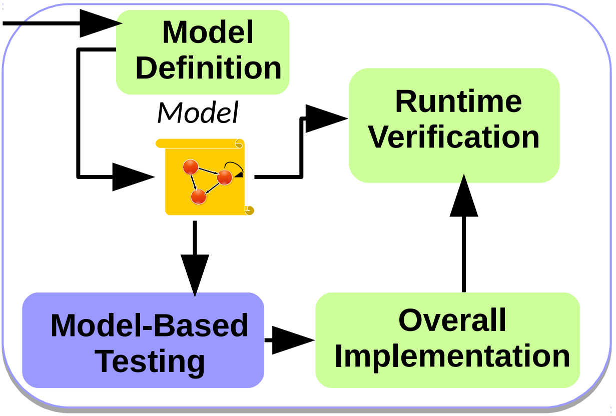
Runtime Verification

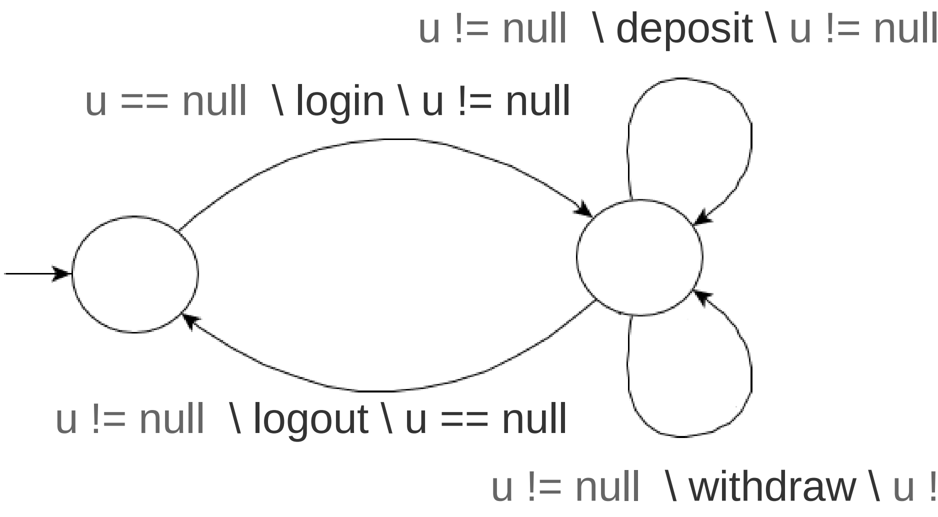
Model-Based Testing

Overall Implementation

Example

- Use MBT to develop the methods



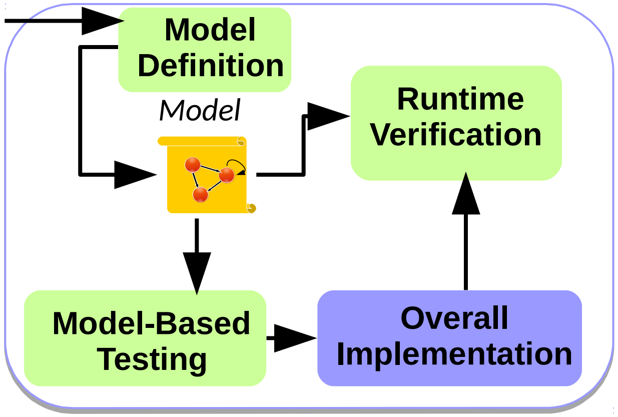

u != null \ deposit \ u != null
u == null \ login \ u != null
u != null \ logout \ u == null
u != null \ withdraw \ u != null

```
/**
 * Deposits money in user's account.
 *
 * @param money amount of money to deposit
 */
public void deposit(int money){
     if (u != null)
         u.getAccount().deposit(money);
}
```

Example

- Finish the overall implementation

(i.e. implement method *main)*

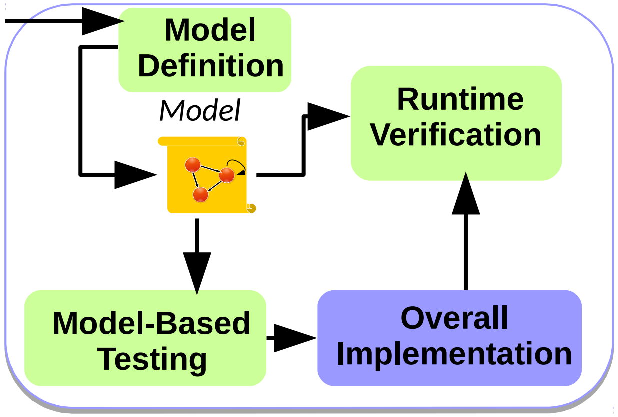


```
switch (inputLine) {
        case "deposit":
                System.out.print("Enter amount to deposit: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
        case "withdraw":
                System.out.print("Enter amount to withdraw: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
```

Example

- Finish the overall implementation

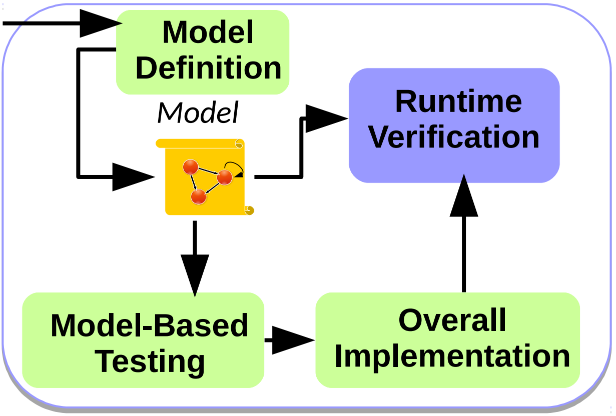
(i.e. implement method *main)*



```
switch (inputLine) {
        case "deposit":
                System.out.print("Enter amount to deposit: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
        case "withdraw":
                System.out.print("Enter amount to withdraw: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
```

Example

- Use runtime verification to validate the model

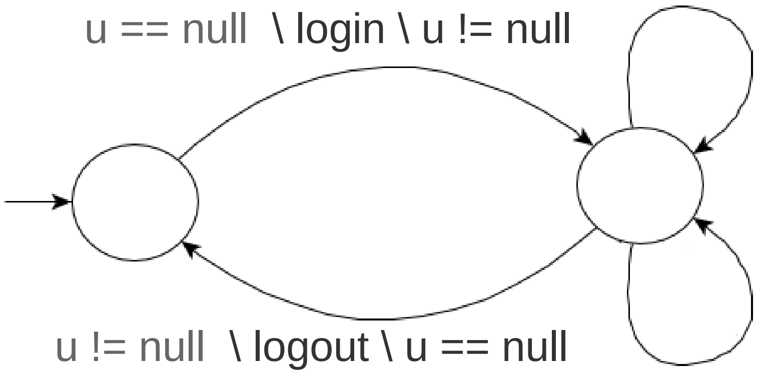

Model Definition

Model

Runtime Verification

Model-Based Testing

Overall Implementation


u != null \ deposit \ u != null

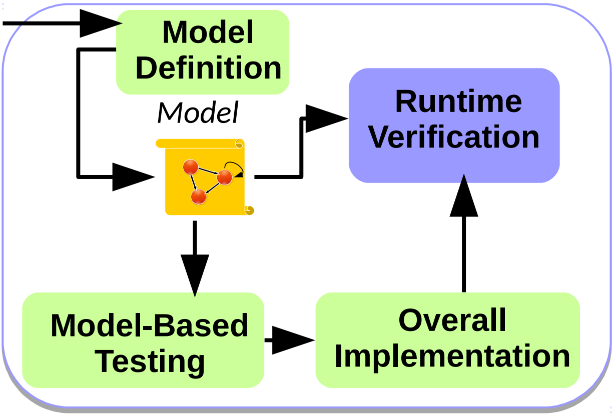
u == null \ login \ u != null

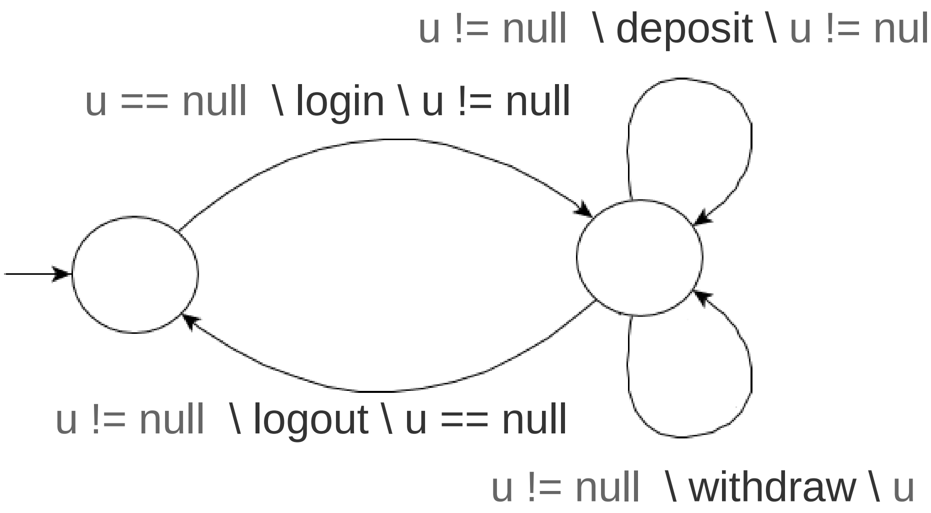
u != null \ logout \ u == null

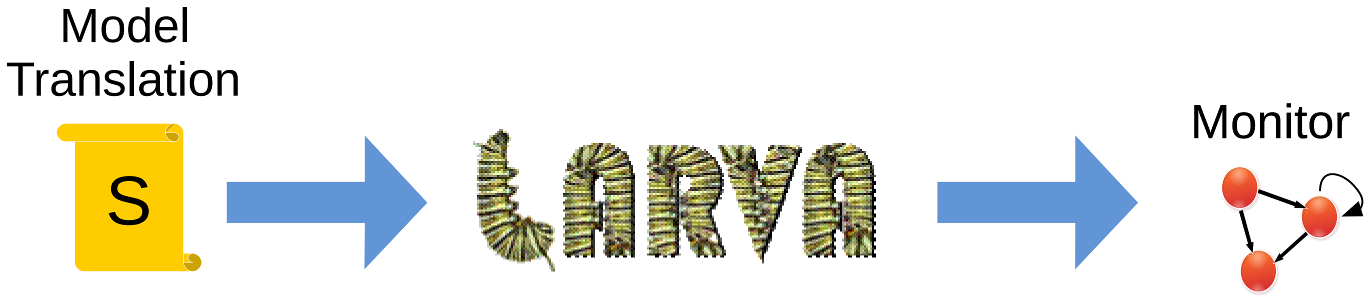
u != null \ withdraw \ u != null

Example

- Use runtime verification to validate the model

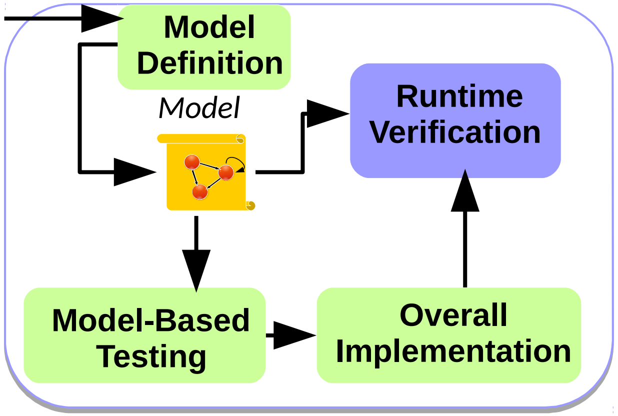



u != null \ deposit \ u != null
u == null \ login \ u != null
u != null \ logout \ u == null
u != null \ withdraw \ u != null



Example

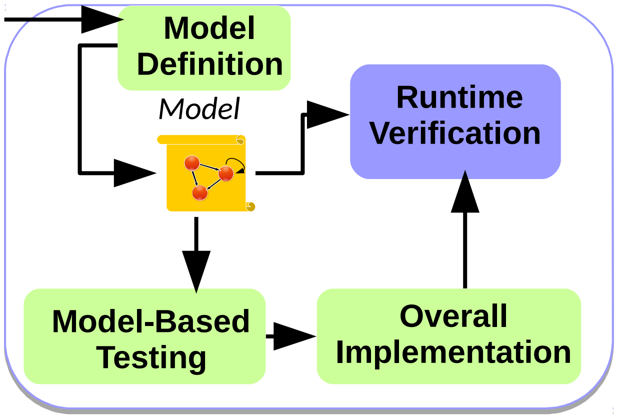
- Execute the monitor against MBT traces



```
switch (inputLine) {
        case "deposit":
                System.out.print("Enter amount to deposit: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
        case "withdraw":
                System.out.print("Enter amount to withdraw: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
```

Example

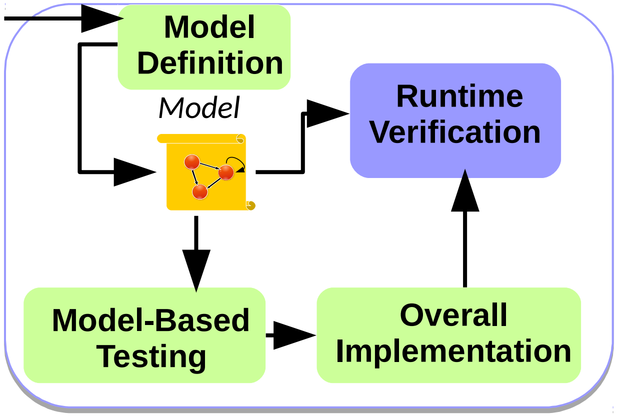
- Execute the monitor against MBT traces



```
switch (inputLine) {
        case "deposit":
                System.out.print("Enter amount to deposit: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.deposit(aux);
                break;
        case "withdraw":
                System.out.print("Enter amount to withdraw: ");
                amount = in.next();
                aux = Integer.parseInt(amount);
                f.withdraw(aux);
                break;
```

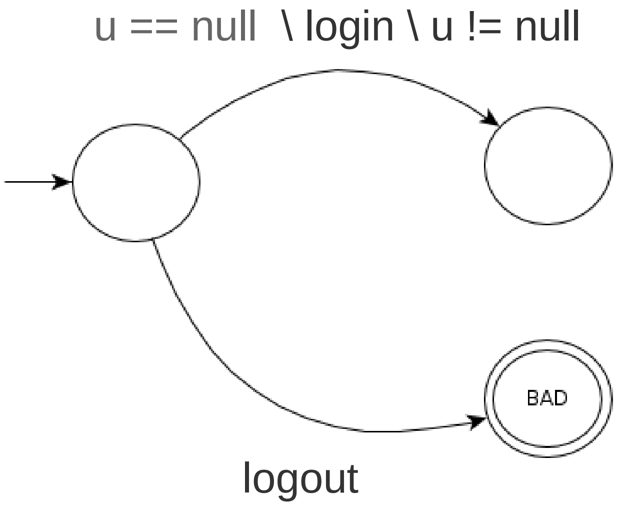
Example

- Extending the monitor



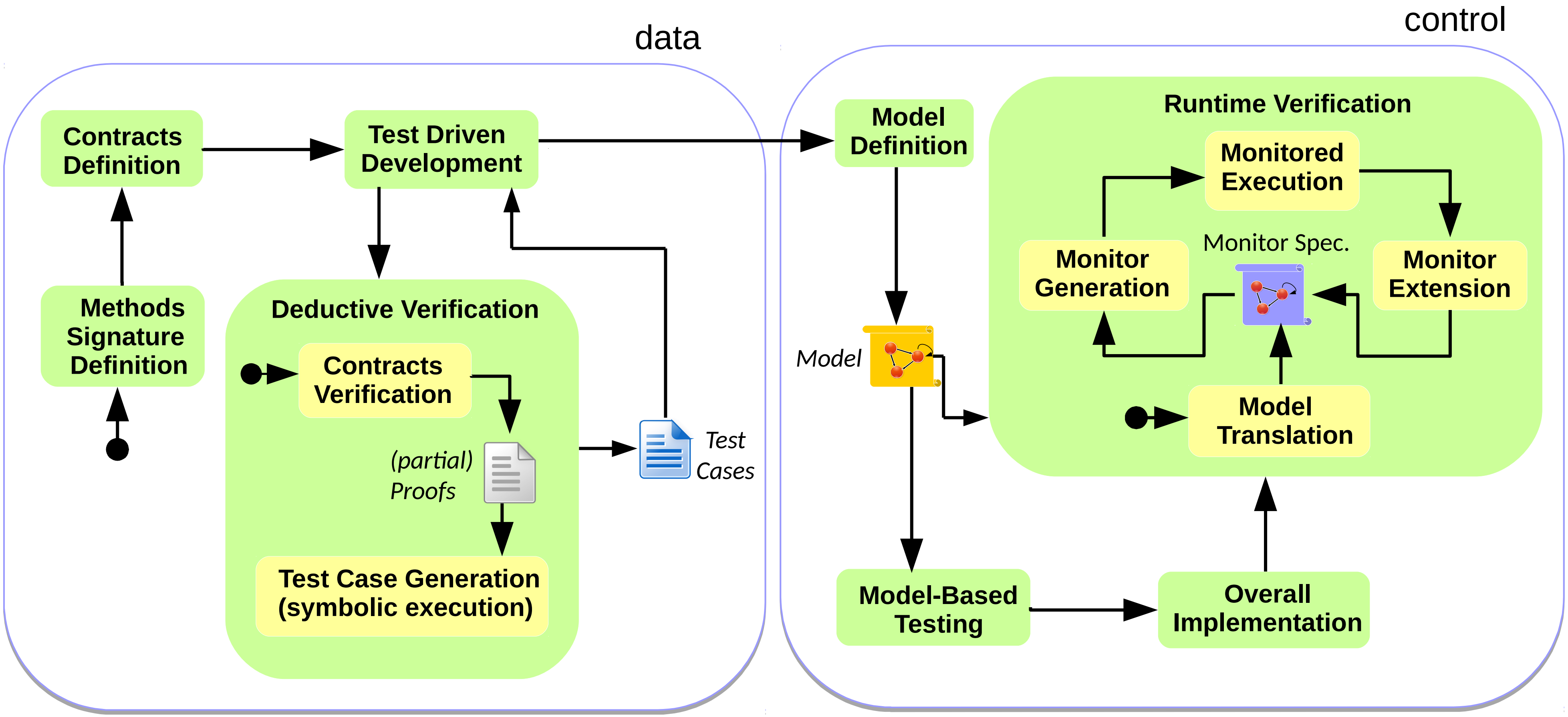
“safety”

data integrity

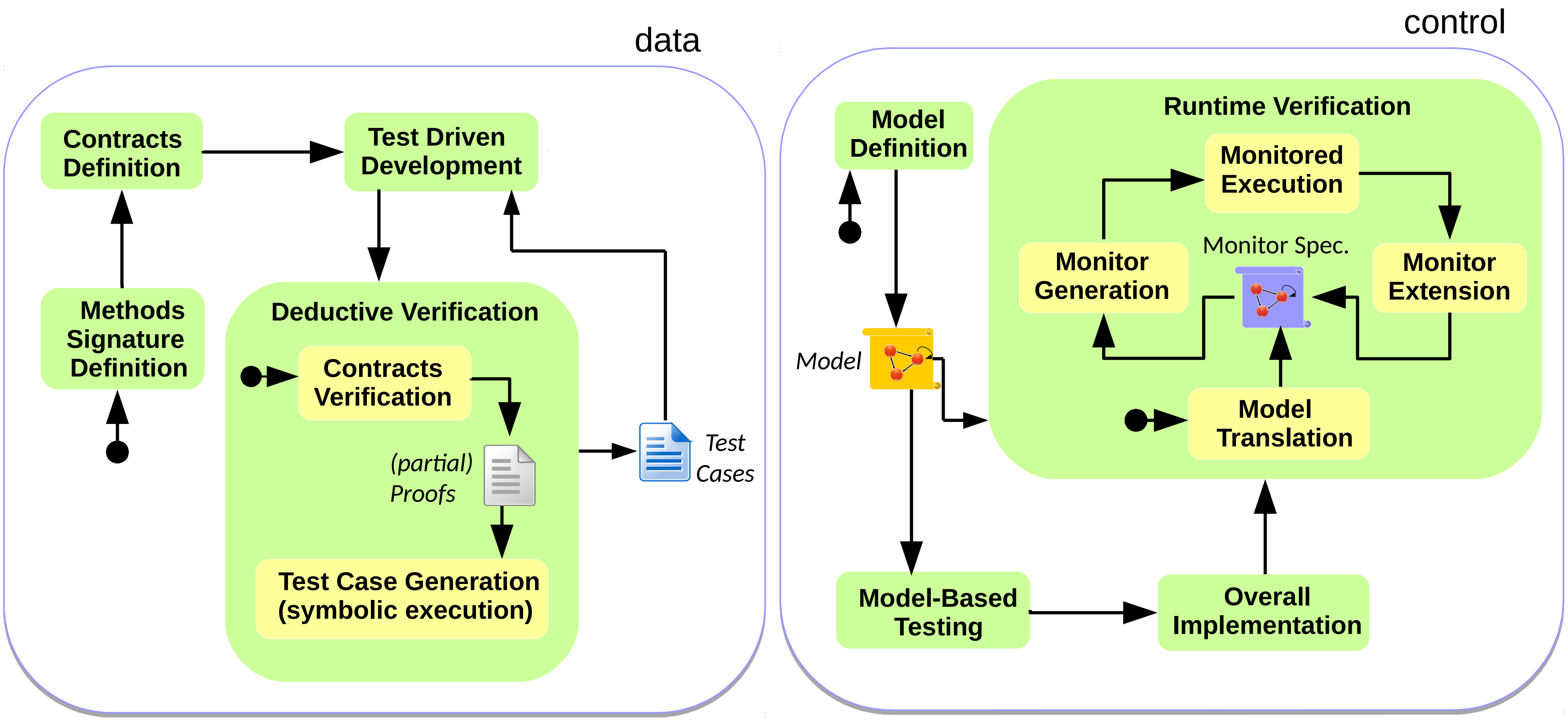


```
public void deposit(int money){
    if (u != null)
        u.getAccount().deposit(money);
}
```

Testing Meets
Deductive and Runtime Verification



Usage Remarks



data

Contracts Definition

Test Driven Development

Methods Signature Definition

Deductive Verification

Contracts Verification

(partial) Proofs

Test Case Generation (symbolic execution)

Test Cases

control

Model Definition

Model

Runtime Verification

Monitored Execution

Monitor Generation

Monitor Spec.

Monitor Extension

Model Translation

Model-Based Testing

Overall Implementation

Conclusions

- Test focus development technique enhanced with formal verification
- (Static) deductive verification enhances TDD when dealing with data aspects
- Runtime Verification enhances MBT when dealing with control aspects
- Compositional usage of the different parts of the proposed technique