Lecture 10
Design by Contract

Zvonimir Rakamarić
University of Utah

Design by Contract

- Also called assume-guarantee reasoning
- Developers annotate software components with contracts (formal specifications)
  - Document developer’s intent
  - Complex system verification broken down into compositional verification of each component
- Typical contracts
  - Annotations on procedure boundaries
    - Preconditions
    - Postconditions
  - Annotations on loop boundaries
    - Loop invariants

Design by Contract cont.

- First used in Eiffel [Bertrand Meyer]
- Native support:
  - Eiffel, Racket, SPARK Ada, Spec#, Dafny,…
- Third-party support:
  - Frama-C
  - Code Contracts project for .NET
  - Java Modeling Language
  - Contracts for Python (PyContracts)
  - contracts.ruby
  - …
- Runtime or static checking of contracts

Design by Contract cont.

- Used when developing high assurance systems for rigorous testing, documentation, and verification
  - Avionics
  - Cars
  - Space
  - Railways
- Used when developing traditional software to formally and conveniently write specifications
  - Turned into assertions during runtime

Code Contracts Example

```
static int BinarySearch(int[] array, int value)
{
  Contract.Requires(array != null);
  …
}
```

PyContracts

```
@contract
def foo(a:'int,>0', b:'list[N],N>0') -> 'list[N]':
  # Requires b to be a nonempty list,
  # and the return value to have the
  # same length
…
```

Spec# Example

```
static int BinarySearch(int[] a, int key)
requires forall{int i in (0: a.Length), int j in
  (i: a.Length); a[i] <= a[j]};
ensures 0 <= result ==> a[result] == key;
ensures result < 0 ==> forall{int i in (0:
  a.Length); a[i] != key};
{
  …
}
```

Java Modeling Languge (JML) Example

```
class BankingExample {
  public static final int MAX_BAL = 1000;
  private int balance;
  //@ invariant balance >= 0 && balance <= MAX_BAL;

  //@ ensures balance == 0;
  public BankingExample() { this.balance = 0; }

  //@ requires 0 < amount && amount+balance < MAX_BAL;
  //@ ensures balance == \old(balance) + amount;
  public void credit(int amount) {
    this.balance += amount;
  }
}
```

Assume-Guarantee Reasoning

- Example

```
foo() {…}
bar() {…foo();…}
```

- How to verify/check bar?

Assume-Guarantee Reasoning cont.

- Solution 1
  - Inline foo
- Solution 2
  - Write contract/specification P of foo
  - Assume P when checking bar

    bar() {…assume P;…}
  - Guarantee P when checking foo

    foo() {…assert P;}
- Pros/cons?

Procedure

```
procedure M(x,y,z) returns (r,s,t)
requires P
ensures Q
{S}
```

- requires is a precondition
  - Predicate P has to hold at procedure entry
- ensures is a postcondition
  - Predicate Q has to hold at procedure exit
- S is procedure body

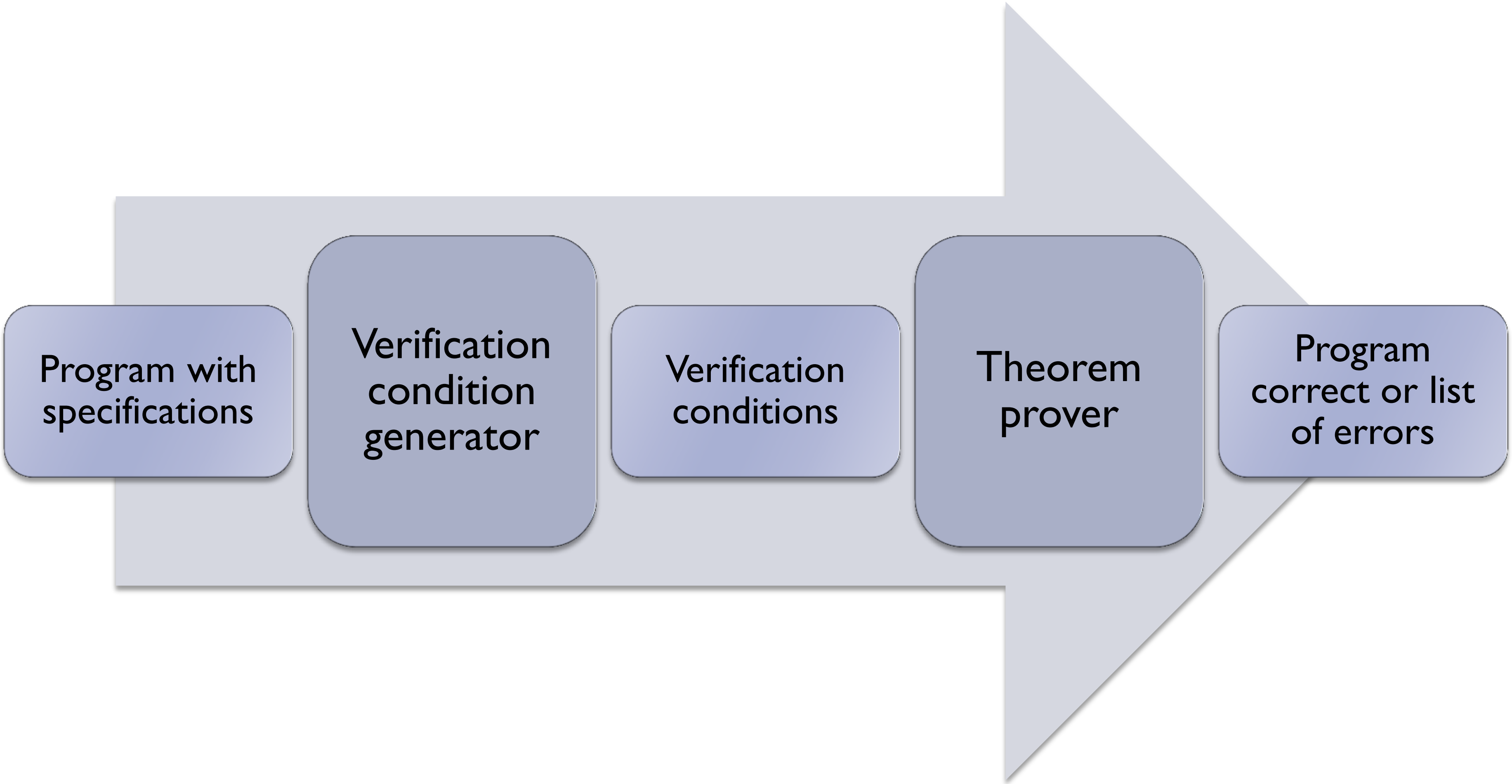
Procedure Example

```
procedure abs(x) returns (abs_x)
requires -1000 < x && x < 1000
ensures abs_x >= 0
{
  if (x >= 0) {
    abs_x := x;
  } else {
    abs_x := -x;
  }
}
```

Dafny

- Simple “verifying compiler”
  - Proves procedure contracts statically for all possible inputs
  - Uses theory of weakest preconditions
- Input
  - Annotated program written in simple imperative language
    - Preconditions
    - Postconditions
    - Loop invariants
- Output
  - Correct or list of failed annotations

Dafny Architecture



Program with specifications

Verification condition generator

Verification conditions

Theorem prover

Program correct or list of errors

Exercise 1

```
procedure abs(x) returns (abs_x)
requires -1000 < x && x < 1000
ensures abs_x >= 0
{
  if (x >= 0) {
    abs_x := x;
  } else {
    abs_x := -x;
  }
}
```

Exercise 2

- Write a method Max that takes two integer parameters and returns their maximum
- Add appropriate annotations and make sure your code verifies

Exercise 3

- Write a test method that calls your Max method from Exercise 2 and then asserts something about the result

While Loop with Invariant

```
while E
  invariant J
do
  S
end
```

loop condition

loop invariant

loop body

- Loop body S executed as long as loop condition E holds
- Loop invariant J must hold on every iteration
  - J must hold initially and is evaluated before E
  - J must hold even on final iteration when E is false
  - Provided by a user or inferred automatically

Exercise 4

```
method m(n: int)
{
    var i := 0;
    while i < n
        invariant 0 <= i
    {
        i := i + 1;
    }
    assert i == n;
}
```

(Dumb) Example: Multiply by 2

```
method Multiply2(n:int) returns (r:int)
{
  r := 0;
  var i:int := 0;
  while (i < n)
  {
    r := r + 2;
    i := i + 1;
  }
}
```

- Specification:
  - Given a non-negative integer **`n`**, function **`Multiply2`** multiplies it by 2

Arrays in Dafny

var a: array<int>

- Can be null
- Have a built in length field (a.Length)
- All array accesses must be proven to be within bounds

Simple Array Example

```
method CreateArray(length:int)
  requires length >= 0;
{
  var a := new int[length];
  a[5] := 10;
}
```

Modifies Annotations

- Dafny has to know what heap locations every procedure modifies
    - Related to how proof is being constructed
- Modifies annotations are used for that
- Simple example:

```
method InitializeArray(a:array<int>, e:int)
modifies a;
{
  var i := 0;
  while (i < a.Length) {
    a[i] := e;
    i := i + 1;
  }
}
```

Quantifiers

- Forall
- Exists

Example: Initialize Array

- Signature:

```
InitializeArray(a:array<int>, e:int)
```

- Specification:
  - Initializes elements of array **`a`** to **`e`**

Example: Linear Search

- Signature:

```
LinearSearch(a:array<int>, l:int,
u:int, e:int) returns (r:bool)
```

- Specification:
  - Returns `true` if `e` is found in array `a` between `l` and `u`, otherwise returns `false`

Useful Links

- https://en.wikipedia.org/wiki/Design_by_contract
- http://rise4fun.com/Dafny/tutorial/guide
- http://research.microsoft.com/en-us/projects/dafny/
- https://www.youtube.com/watch?v=spcfzbisBv4
- http://research.microsoft.com/en-us/projects/contracts/
- https://pypi.python.org/pypi/PyContracts