

# Automated Structural Testing with PathCrawler. Tutorial for QSIC 2012.

## Examples.

*Nicky.WILLIAMS@cea.fr, Nikolai.KOSMATOV@cea.fr*  
*CEA, LIST, Software Safety Lab*  
*Saclay (Paris), France*

*Xi'an, 27th August, 2012*

### Example 1. Function under test: **Tritype**

```
/* Should return the type of the triangle
   which has sides of these lengths.
   3 = not a triangle
   2 = equilateral triangle
   1 = isocèles triangle
   0 = other triangle
*/
int Tritype(double i, double j, double k){
    int trityp = 0;
    if (i < 0.0 || j < 0.0 || k < 0.0)           // line 10
        return 3;
    if (i + j <= k || j + k <= i || k + i <= j) // line 12
        return 3;
    if (i == j) trityp = trityp + 1;             // line 14
    if (i == k) trityp = trityp + 1;             // line 15
    if (j == k) trityp = trityp + 1;             // line 16
    if (trityp >= 2)                             // line 17
        trityp = 2;
    return trityp;
}
```

## Default test parameters:

### Customization of Test Parameters

The generator will use values for these inputs, chosen from these domains, to construct the test-cases. You can restrict the interval to ensure that the test-cases are pertinent. You can also remove inputs which you know are never referenced by the tested function. The addition of new inputs can be used for inputs you removed by mistake or to add specific array elements in order to give them a different domain from that of all the other elements of the same array.

Confirm parameters

#### Domains of input array elements

#### Domains of input variables

 -1.7976931348 <= k <= 1.7976931348

 -1.7976931348 <= j <= 1.7976931348

 -1.7976931348 <= i <= 1.7976931348

#### Simple preconditions (inequalities)

#### Quantified preconditions

#### Path selection strategy

all

#### C precondition was not found

## Example 2. Function under test: **Tritype**

```
/* Should return the type of the triangle
   which has sides of these lengths.

   2 = equilateral triangle
   1 = isoceses triangle
   0 = other triangle
*/
int Tritype(double i, double j, double k){
    int trityp = 0;
    // if (i < 0.0 || j < 0.0 || k < 0.0)           // line 10
    //     return 3;
    // if (i + j <= k || j + k <= i || k + i <= j)//line 12
    //     return 3;
    if (i == j) trityp = trityp + 1;                // line 14
    if (i == k) trityp = trityp + 1;                // line 15
    if (j == k) trityp = trityp + 1;                // line 16
    if (trityp >= 2)                                // line 17
        trityp = 2;
    return trityp;
}
```

**Default test parameters: see Example 1**

## Example 4. Function under test: **Tritype**

```
/* Should return the type of the triangle
   which has sides of these lengths.

   2 = equilateral triangle
   1 = isoceles triangle
   0 = other triangle
*/
int Tritype(double i, double j, double k){
    int trityp = 0;
    // if (i < 0.0 || j < 0.0 || k < 0.0)           // line 10
    //     return 3;
    // if (i + j <= k || j + k <= i || k + i <= j)//line 12
    //     return 3;
    if (i == j) trityp = trityp + 1;                // line 14
    if (i == k) trityp = trityp + 1;                // line 15
    if (j == k) trityp = trityp + 1;                // line 16
    if (trityp >= 2)                                // line 17
        trityp = 2;
    return trityp;
}

int Tritype_precond(double i, double j, double k){
    if (i < 0.0 || j < 0.0 || k < 0.0)             // line 23
        return 0;
    if (i + j <= k || j + k <= i || k + i <= j) // line 25
        return 0;
    return 1;
}
```

## Default test parameters:




### Customization of Test Parameters

The generator will use values for these inputs, chosen from these domains, to construct the test-cases. You can restrict the interval to ensure that the test-cases are pertinent. You can also remove inputs which you know are never referenced by the tested function. The addition of new inputs can be used for inputs you removed by mistake or to add specific array elements in order to give them a different domain from that of all the other elements of the same array.

Confirm parameters

#### Domains of input array elements

#### Domains of input variables

 -1.7976931348 <= k <= 1.7976931348  
 -1.7976931348 <= j <= 1.7976931348  
 -1.7976931348 <= i <= 1.7976931348

#### Simple preconditions (inequalities)

#### Quantified preconditions

#### Path selection strategy

all

#### C precondition was found. You may activate or ignore it.

pathcrawler\_\_Tritype\_precond  activate

## Example 5.

### Function under test: Merge

```
/* Should copy all the elements
   of ordered arrays t1 and t2
   into the ordered array t3 */

void Merge(int t1[3], int t2[3], int t3[6]) {

    int i = 0, j = 0, k = 0 ;

    while (i < 3 && j < 3) { // line 09
        if (t1[i] < t2[j]) { // line 10
            t3[k] = t1[i];
            i++;
        }
        else {
            t3[k] = t2[j];
            j++;
        }
        k++ ;
    }
    while (i < 3) { // line 20
        t3[k] = t1[i];
        i++;
        k++;
    }
    while (j < 3) { // line 25
        t3[k] = t2[j];
        j++;
        k++;
    }
}
```


## Default test parameters:


### Customization of Test Parameters


The generator will use values for these inputs, chosen from these domains, to construct the test-cases. You can restrict the interval to ensure that the test-cases are pertinent. You can also remove inputs which you know are never referenced by the tested function. The addition of new inputs can be used for inputs you removed by mistake or to add specific array elements in order to give them a different domain from that of all the other elements of the same array.

Confirm parameters


#### Domains of input array elements


  <=  <=


  <=  <=

  <=  <=

#### Domains of input variables

  <=  <=

  <=  <=

  <=  <=

#### Simple preconditions (inequalities)

#### Quantified preconditions

#### Path selection strategy

## Example 7.

### Function under test: Merge

```
/* Should copy all the elements
   of ordered arrays t1 and t2
   of lengths l1 and l2
   into the ordered array t3 */
void Merge (int t1[], int t2[], int t3[], int l1, int l2) {


    int i = 0, j = 0, k = 0 ;


    while (i < l1 && j < l2) { // line 09
        if (t1[i] < t2[j]) { // line 10
            t3[k] = t1[i];
            i++;
        }
        else {
            t3[k] = t2[j];
            j++;
        }
        k++;
    }
    while (i < l1) { // line 20
        t3[k] = t1[i];
        i++;
    }
    while (j < l2) { // line 25
        t3[k] = t2[j];
        j++;
        k++;
    }
}
```




## Predefined test parameters:


### Domains of input array elements

 -100 <= t2[INDEX\_0] <= 100

 -100 <= t1[INDEX\_0] <= 100


### Domains of input variables

 -2147483648 <= l2 <= 2147483647

 -2147483648 <= l1 <= 2147483647

 0 <= dim(t3) <= 1


 0 <= dim(t2) <= 1


 0 <= dim(t1) <= 1


### Simple preconditions (inequalities)

4


### Quantified preconditions


 for all INDEX


such that: 

INDEX < dim(t1)-1 

we have: t1[INDEX] <= t1[INDEX+1]

 for all INDEX

such that: 

INDEX < dim(t2)-1 

we have: t2[INDEX] <= t2[INDEX+1]

### Path selection strategy

all

# Example 10a. Function under test: Merge (see Example 7), Predefined test parameters:

**Domains of input array elements** ⓘ

<=  <=

<=  <=

---

**Domains of input variables** ⓘ

<=  <=

<=  <=

<=  <=

<=  <=

<=  <=

---

**Simple preconditions (inequalities)** ⬇

==

==

==

---

**Quantified preconditions** ⬇

such that:  <

we have:  <=

---

such that:  <

we have:  <=

---

**Path selection strategy** ⓘ

---

**C precondition was not found** ⓘ

## Oracle:

```
void oracle_Merge(
    int Pre_t1[], int t1[],
    int Pre_t2[], int t2[],
    int Pre_t3[], int t3[],
    int Pre_l1, int l1,
    int Pre_l2, int l2)
{
    int i, j, n1, n2, n3;
    int l3 = l1 + l2;
    int l3moins1 = l3 -1;

    for (i = 0; i < l3moins1; i++) {
        if (t3[i] > t3[i+1]) {
            pathcrawler_verdict_failure(); /* t3 not ordered */
            return;
        }
    }
    pathcrawler_verdict_success();
}
```

## Example 10b. Function under test: Merge (see Example 7)

Predefined test parameters: see Example 10a.

### Oracle:

```
void oracle_Merge(
    int Pre_t1[], int t1[],
    int Pre_t2[], int t2[],
    int Pre_t3[], int t3[],
    int Pre_l1, int l1,
    int Pre_l2, int l2)
{
    int i, j, n1, n2, n3;
    int l3 = l1 + l2;
    int l3moins1 = l3 -1;

    for (i = 0; i < l3moins1; i++) {
        if (t3[i] > t3[i+1]) {
            pathcrawler_verdict_failure(); /* t3 not ordered */
            return;
        }
    }
    i = 0;
    while (i < l3) {
        /* count occurrences of this element in t3 */
        n3 = 1;
        while (i < l3moins1 && t3[i + 1] == t3[i]) {
            i++;
            n3++;
        }
        /* count occurrences of this element in t1 */
        n1 = 0;
        for (j = 0; j < l1; j++) {
            if (t1[j] == t3[i])
                n1++;
        }
        /* count occurrences of this element in t2 */
        n2 = 0;
        for (j = 0; j < l2; j++) {
            if (t2[j] == t3[i])
                n2++;
        }
        /* compare */
        if (n3 != (n1 + n2)) {
            pathcrawler_verdict_failure(); /* t3 does not have the correct
number of occurrences of all elements */
            return;
        }
        i++;
    }
    pathcrawler_verdict_success();
}
```

## Example 10c. Function under test: Merge (see Example 7)

Predefined test parameters: see Example 10a.

### Oracle:

```
void oracle_Merge(
    int Pre_t1[], int t1[],
    int Pre_t2[], int t2[],
    int Pre_t3[], int t3[],
    int Pre_l1, int l1,
    int Pre_l2, int l2)
{
    int i, j, n1, n2, n3;
    int l3 = l1 + l2;
    int l3moins1 = l3 -1;

    for (i = 0; i < l1; i++) {
        if (Pre_t1[i] != t1[i]) {
            pathcrawler_verdict_failure(); /* t1 modified */
            return;
        }
    }

    for (i = 0; i < l2; i++) {
        if (Pre_t2[i] != t2[i]) {
            pathcrawler_verdict_failure(); /* t2 modified */
            return;
        }
    }

    for (i = 0; i < l3moins1; i++) {
        if (t3[i] > t3[i+1]) {
            pathcrawler_verdict_failure(); /* t3 not ordered */
            return;
        }
    }
    i = 0;
    while (i < l3) {
        /* count occurrences of this element in t3 */
        n3 = 1;
        while (i < l3moins1 && t3[i + 1] == t3[i]) {
            i++;
            n3++;
        }
        /* count occurrences of this element in t1 */
        n1 = 0;
        for (j = 0; j < l1; j++) {
            if (t1[j] == t3[i])
                n1++;
        }
        /* count occurrences of this element in t2 */
        n2 = 0;
        for (j = 0; j < l2; j++) {
            if (t2[j] == t3[i])
                n2++;
        }
        /* compare */
        if (n3 != (n1 + n2)) {
            pathcrawler_verdict_failure(); /* t3 does not have the correct number of occurrences of all
elements */
            return;
        }
        i++;
    }
    pathcrawler_verdict_success();
}
```

## Example Uninit. Function under test: `uninit_var`

```
int uninit_var(int a[3], int b[3]) {
    int i, k;
    for(i=0; i<2; i++) {          // line 3
        if(a[i] == 0)             // line 4
            return 0;
        if(a[i] != a[i+1])       // line 6
            k = 0;
        else
            if(k == 2)            // line 9
                return 0;
        while(b[k] != a[i])       // line 11
            if(k == 2)           // line 12
                return 0;
            else
                k++;
    }
    return 1;
}
```

## Example UC. Function under test: `Bsearch`

```
/* Should return 1 if x is an element of ordered array A */
int Bsearch( int A[8], int x) // tested function
{
    int low, high, mid, found ;

    low = 0 ;
    high = 7 ;
    found = 0 ;
    while( high > low )          // line 09
        { mid = (low + high) / 2 ;

            if( x == A[mid] )    // line 12
                found = 1;


            if( x > A[mid] )     // line 15
                low = mid + 1 ;
            else
                high = mid - 1;
        }
    mid = (low + high) / 2 ;

    if( ( found != 1) && ( x == A[mid] ) ) // line 22
        found = 1;

    return found ;
}
```

## Predefined test parameters:

### Domains of input array elements

 -100 <= A[INDEX\_0] <= 100




### Domains of input variables

 -100 <= x <= 100

 8 <= dim(A) <= 8

### Simple preconditions (inequalities)

### Quantified preconditions

 for all INDEX  
such that:   
INDEX < 7   
we have: A[INDEX] <= A[INDEX+1]

### Path selection strategy

all

### C precondition was not found

### Oracle function in C language

```
void oracle_Bsearch(
  int *Pre_A, int *A,
  int Pre_x, int x,
  int pathcrawler, retres Bsearch)
{
  int i;
```

## Oracle:

```
void oracle_Bsearch(
    int *Pre_A, int *A,
    int Pre_x, int x,
    int pathcrawler__retres__Bsearch)
{
    int i;
    int present = 0;
    int modif = 8;

    for(i=0; i<8; i++) {
        if(A[i] != Pre_A[i])
            modif = i;
        if(Pre_A[i] == Pre_x)
            present = 1;
    }
    if(present==0 && present != pathcrawler__retres__Bsearch) {
        pathcrawler_verdict_failure(); } /* implementation wrongly
found x in A */
    else {
        if(present==1 && present != pathcrawler__retres__Bsearch) {
            pathcrawler_verdict_failure(); } /* implementation wrongly
said x was not in A */
        else {
            if(modif < 8) {
                pathcrawler_verdict_failure(); } /* implementation modified A
*/
            else {pathcrawler_verdict_success();
                }
            }
        }
    }
    return;
}
```



## Example Chance. Function under test: Bsearch

```
/* Should return 1 if x is an element of ordered array A */
int Bsearch( int A[10], int x) // tested function
{
    int low, high, mid, found ;

    low = 0 ;
    high = 9 ;
    found = 0 ;
    while( high > low ) // line 09
        { mid = (low + high) / 2 ;

            if( x == A[mid] ) // line 12
                found = 1;

            if( x > A[mid] ) // line 15
                low = mid + 1 ;
            else
                high = mid - 1;
        }
    mid = (low + high) / 2 ;

    if( ( found != 1) && ( x >= A[mid]) ) // line 22
        found = 1;

    return found ;
}
```

### Example 11. Function under test: Bsearch

```
/* Should return 1 if x is an element of ordered array A */
int Bsearch( int A[8], int x) // tested function
{
    int low, high, mid, found ;

    low = 1 ;
    high = 7 ;
    found = 0 ;
    while( high > low ) // line 09
        { mid = (low + high) / 2 ;

            if( x == A[mid] ) // line 12
                found = 1;

            if( x > A[mid] ) // line 15
                low = mid + 1 ;
            else
                high = mid - 1;
        }
    mid = (low + high) / 2 ;

    if( ( found != 1) && ( x == A[mid]) ) // line 22
        found = 1;

    return found ;
}
```

**Predefined test parameters and oracle: see Example UC**

## Example 12. Function under test: CompareBsearchSpec

```
/* Should return 1 if x is an element of ordered array A */
int Bsearch( int A[8], int x)
{
    int low, high, mid, found ;

    low = 1 ;
    high = 7 ;
    found = 0 ;
    while( high > low ) // line 09
        { mid = (low + high) / 2 ;

            if( x == A[mid] ) // line 12
                found = 1;

            if( x > A[mid] ) // line 15
                low = mid + 1 ;
            else
                high = mid - 1;
        }
    mid = (low + high) / 2 ;

    if( ( found != 1) && ( x == A[mid]) ) // line 22
        found = 1;

    return found ;
}

int spec_Bsearch(
    int *Pre_A, int *A,
    int Pre_x, int x,
    int result_implementation)
{
    int i;
    int present = 0;
    int modif = 8;

    for(i=0; i<8; i++) // line 38
        {
            if(A[i] != Pre_A[i]) // line 40
                modif = i;
            if(Pre_A[i] == Pre_x) // line 42
                present = 1;
        }
    if(present==0 && present != result_implementation) {
        return 0; } /* implementation wrongly found x in A */
    else {
        if(present==1 && present != result_implementation) {
            return 0; } /* implementation wrongly said x was not in A */
        else {
            if(modif < 8) { // line 51
                return 0; } /* implementation modified A */
            else return 1;
        }
    }
    return;
}

int CompareBsearchSpec(int A[8], int x) // tested function
{
    int *Pre_A = (int *)malloc(8 * sizeof(int));
    int i;
    for (i = 0; i < 8; i++) // line 64
        Pre_A[i] = A[i];
    int result_implementation=Bsearch(A,x);
    return(spec_Bsearch(Pre_A,A,x,x,result_implementation));
}
```

**Predefined test parameters: see Example UC**

**Oracle:**

```
void oracle_CompareBsearchSpec(
    int *Pre_A, int *A,
    int Pre_x, int x,
    int pathcrawler__retres__CompareBsearchSpec)
{
    if (pathcrawler__retres__CompareBsearchSpec) {
        pathcrawler_verdict_success(); }
    else {
        pathcrawler_verdict_failure(); }
    return;
}
```

**Example 13. Function under test: f**

```
int x, y;
int f ( int a ) {
    int sum;
    if(a == 0){ // line 04
        x = 0; y = 0;
    }else{
        x = 5; y = 5;
    }
    sum = x + y; // sum can be 0
    pathcrawler_assert( sum != 0 );
    return 10 / sum; // risk of division by 0
}
```

**Example 14. Function under test: f**

```
int x, y;
int f ( int a ) {
    int sum;
    if(a == 0){ // line 04
        x = 0; y = 5;
    }else{
        x = 5; y = 0;
    }
    sum = x + y; // sum cannot be 0
    pathcrawler_assert( sum != 0 );
    return 10 / sum; // risk of division by 0
}
```

### Example 15. Function under test: `ArrayCmp`

```
/* compares (with respect to the lexicographic order) the subarrays
with the first n elements of given arrays t1 and t2,
returns 0 if the subarrays are equal,
      1 if the subarray in t1 is greater than in t2,
     -1 if the subarray in t2 is greater than in t1
*/

int ArrayCmp(int n, int* t1, int* t2) {
    int i;
    for (i = 0; i <= n; i++) {    /* line 10 */
        if (t1[i] > t2[i])        /* line 11 */
            return -1;
        else if (t1[i] < t2[i])   /* line 13 */
            return 1;
    }
    return 0;
}
```

### Example 16. Function under test: `ArrayCmp`

```
/* compares (with respect to the lexicographic order) the subarrays
with the first n elements of given arrays t1 and t2,
returns 0 if the subarrays are equal,
      1 if the subarray in t1 is greater than in t2,
     -1 if the subarray in t2 is greater than in t1
*/

int ArrayCmp(int n, int* t1, int* t2) {
    int i;
    for (i = 0; i < n; i++) {    /* line 10 */
        if (t1[i] > t2[i])        /* line 11 */
            return -1;
        else if (t1[i] < t2[i])   /* line 13 */
            return 1;
    }
    return 0;
}
```

## Example 17. Function under test: bsort

```
/* Bubble sort of a given array 'table' of
   a given length 'l' in ascending order */

void bsort (int * table, int l)
{
  int i, temp, nb;
  char done;
  done = 0;
  nb = 0;
  while ( !done && (nb < l-1)){          /* line 10 */
    done = 1;
    for (i=0 ; i<l-1 ; i++)            /* line 12 */
      if (table[i] > table[i+1]){      /* line 13 */
        done = 0;
        temp = table[i];
        table[i + 1] = temp;
      }
    nb++;
  }
}
```