Loop invariants and some more

12. January 2011

Jens Gerlach

Fraunhofer FIRST

Some astyle options

- please use these options!

```
astyle -A1 -j -f -p -Y -z2 --indent=spaces
```

- for more details see

  http://astyle.sourceforge.net/astyle.html

An abstract discussion of loop invariants

An abstract example

```
/*@
    requires precondition;

    assigns somewhere;

    ensures postcondition;
*/
void foo(int* a, int n)
{
    for(init; cond; step)
    {
        some statements;
    }
}
```

```
/*@
    requires precondition;

    assigns somewhere;

    ensures postcondition;
*/
void foo(int* a, int n)
{
    //@ assert precondition;

    for(init; cond; step)
    {
        some statements;
    }

    //@ assert postcondition;
}
```

Where does *Inv* hold?

```
void foo(int* a, int n)
{
    //@ assert precondition;
    //@ loop invariant Inv;
    for(init; cond; step)
    {
        some statements;
    }

    //@ assert postcondition;
}
```

After the `init` statement

```
void foo(int* a, int n)
{
    //@ assert precondition;

    init;
    //@ assert Inv;

    //@ loop invariant Inv;
    for(;cond; step)
    {
        some statements;
    }

    //@ assert postcondition;
}
```

After evaluating `cond`

```
void foo(int* a, int n)
{
    //@ assert precondition;

    init;
    //@ assert Inv;

    //@ loop invariant Inv;
    for(;cond; step)
    {
        //@ assert Inv;
        some statements;
    }

    //@ assert postcondition;
}
```

After executing `step`

```
void foo(int* a, int n)
{
    //@ assert precondition;

    init;
    //@ assert Inv;

    //@ loop invariant Inv;
    while(condition)
    {
        //@ assert Inv;
        some statements;
        step;
        //@ assert Inv;
    }

    //@ assert postcondition;
}
```

After leaving the loop

```
void foo(int* a, int n)
{
    //@ assert precondition;

    init;
    //@ assert Inv;

    //@ loop invariant Inv;
    while(condition)
    {
        //@ assert Inv;
        some statements;
        step;
        //@ assert Inv;
    }
    //@ assert Inv;

    //@ assert postcondition;
}
```

Ideally we can derive `postcondition` from `Inv`

```
void foo(int* a, int n)
{
    //@ assert precondition;

    //@ loop invariant Inv;
    for(init; cond; step)
    {
        some statements;
    }
    //@ assert Inv;

    //@ assert postcondition;
}
```

Hints

- for a simple function like in the example the loop invariant is inspired by the post condition!
- for more details on loop invariant see Section 2.4.2 of the ACSL description

  http://frama-c.com/download/acsl_1.5.pdf

An concrete discussion of loop invariants

Computing factorial

```
/*@
   requires 0 <= n < INT_MAX;
   requires Factorial(n) <= INT_MAX;

   assigns \nothing;

   ensures \result == Factorial(n);
*/
int factorial(int n);
{
    int c = 1;

    for(int i = 1; i <= n; ++i)
    {
       c = i * c;
    }
    return c;
}
```

What is the postcondition?

```
int factorial(int n);
{
    int c = 1;

    for(int i = 1; i <= n; ++i)
    {
       c = i * c;
    }

    //@ assert c == Factorial(n);
    return c;
}
```

Which loop invariant?

```
int factorial(int n);
{
    int c = 1;
    /*@
       loop invariant Inv;
    */
    for(int i = 1; i <= n; ++i)
    {
       c = i * c;
    }

    //@ assert c == Factorial(n);
    return c;
}
```

Look at loop initialization!

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert Inv;

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       c = i * c;
    }

    //@ assert c == Factorial(n);
    return c;
}
```

Where else to look?

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert Inv;

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert Inv;
       c = i * c;
    }
    //@ assert Inv;

    //@ assert c == Factorial(n);
    return c;
}
```

A first guess

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert c == Factorial(i-1) || c == Factorial(i);
    //  which disjunct is correct?

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert Inv;
       c = i * c;
    }
    //@ assert Inv;

    //@ assert c == Factorial(n);
    return c;
}
```

Look at loop body

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert c == Factorial(i-1) || c == Factorial(i);
    //  which disjunct is correct?

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert c == Factorial(i);
       // this is wrong because we multiply by i later
       c = i * c;
    }
    //@ assert Inv;

    //@ assert c == Factorial(n);
    return c;
}
```

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert c == Factorial(i-1);

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert c == Factorial(i-1);
       c = i * c;
    }
    //@ assert Inv;

    //@ assert c == Factorial(n);
    return c;
}
```

Loop termination

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert c == Factorial(i-1);

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert c == Factorial(i-1);
       c = i * c;
    }
    //@ assert c == Factorial(i-1);
    // i is n+1 by now!

    //@ assert c == Factorial(n);
    return c;
}
```

Compare with post cond.

```
int factorial(int n);
{
    int c = 1;

    int i = 1;
    //@ assert c == Factorial(i-1);

    /*@
        loop invariant Inv;
    */
    for(; i <= n; ++i)
    {
       //@ assert c == Factorial(i-1);
       c = i * c;
    }
    //@ assert c == Factorial(n);


    //@ assert c == Factorial(n);
    return c;
}
```

Now we have a good candidate

```
int factorial(int n)
{
    int c = 1;
    /*@
        loop invariant c == Factorial(i-1);
        // of course we need more
    */
    for(int i = 1; i <= n; ++i)
    {
       c = i * c;
    }
    return c;
}
```

Final result

```
int factorial(int n)
{
    int c = 1;
    /*@
        loop invariant 1 <= i <= n+1;
        loop invariant c == Factorial(i-1);
        loop assigns i, c;
        loop variant n-i;
    */
    for(int i = 1; i <= n; ++i)
    {
       c = i * c;
    }
    return c;
}
```