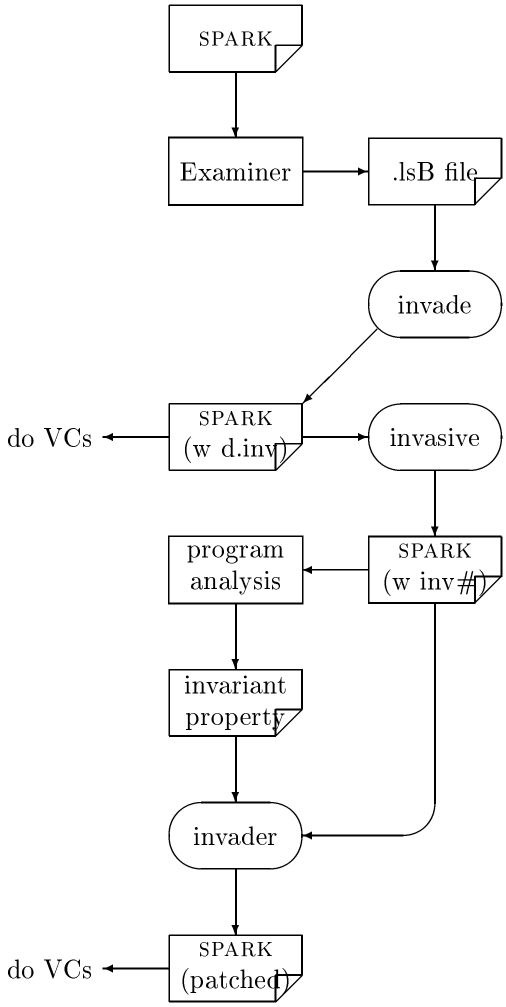
Andrew J. Cook
May 14, 2004

Invariant Patchy

1 Invasion



The invariant patching scripts `invade`, `invasive` and `invader` are for inserting assertion lines into spark source code.

`invade` is an awk script that processes one of the output files from the SPARK examiner(*base*.lsB). It uses the warnings generated by the examiner when a default invariant is added to insert the default invariant (assert true;) into the code.

`invasive` is an awk script that takes the SPARK code with the default invariants added. It adds invariant number annotations to all the invariants it finds in the code. This means that it adds numbers to all invariants regardless of if they were added by the examiner or pre-existing(default or not).

`invader` is a shell script which uses awk first to break down a file containing an invariant property generated by program analysis and second to patch that property into the annotated spark code. The file generated by the program analysis should take the form:

number:*property*

The `invader` script then uses awk script `invaderiser` to search for the assertion annotated with *number* and replace it with an assertion of *property*.

2 Using invaders

2.1 `invade`

The `invade` script takes one argument, the name of the .lsB file and outputs the annotated SPARK to the standard out. So to use it redirect the output into the desired file.

`invade` *input*`.lsB >` *output*

The .lsB file is a result of running the SPARK examiner over the .adb file with the listing option switched on. The .lsB file will contain the source code from the original program with the lines numbered and the warnings interspersing it. `invade` reconstructs the original code from this and uses the invariant warnings to insert a default invariant (`--# assert true;`) into the code.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

2.2 `invasive`

`invasive` takes SPARK code with invariants in it and adds numbering annotations to each invariant to allow the program analyser to uniquly identify each invariant. Like `invade`, `invasive` also writes it's result to the standard output.

`invasive` *input* `>` *output*

In the input file an invariant specified by:

```
--# assert X=1;
```

would be replaced, in the output, by:

`--# assert inv_number(` *n* `) and X=1;`

2.3 `invader`

`invader` is a shell script taking two arguments, the spark code with annotated invariants and the file containing the invariant property from program analysis. The result should be written to the standard output.

`invader` *input property_file* `>` *output*

The *property_ file* should consist of a single line of the form:

n`:`*property*

Where *n* is the invariant number inserted into the code by `invasive` and *property* is the newly discovered invariant property. The line:

`--# assert inv_number(` *n* `) and X=1;`

Would become:

`--# assert` *property*`;`

The `invader` script makes a call to awk using the script `invaderiser`. So `invaderiser` must be present when using the `invader` script.

3 Conclusion

The scripts enable patching invariants into the spark code.

A Example

To illustrate the use of the invaders the `filters` example is shown below as it goes through it's various stages.

A.1 `filters.lsB`

The .lsB file is a result of running the spark examiner over an example. As well as producing vcs and rules it can produce a trace of warnings generated. This trace file reproduces the code(including comments) from the original .adb file so we can use it to reconstruct the source.

```
           *******************************************************
                            Listing of SPARK Text
        SPARK95 Examiner with VC and RTC Generator Release 6.0 / 11.01
                            Demonstration Version
           *******************************************************


                       DATE : 21-APR-2004 16:00:04.32

Line
   1
   2  package body Filter_Sum is
   3
   4     procedure Filter_S(D: in D_Type; R: out
   5     Integer)
   6     is
   7     begin
   8        R:= 0;
   9        for I in I_Type loop

--- (  1)  Warning           :402: Default assertion planted to cut loop.

  10           if D(I) >= 0 and D(I) <= 100 then
  11              R:= R + D(I);
  12           end if;
  13        end loop;
  14     end Filter_S;

+++        Flow analysis of subprogram Filter_S performed:
           no errors found.

  15
  16  end Filter_Sum;


--End of file--------------------------------------------------
```

A.2 `filters.add`

I called the source with default invariants added `filters.add`. The warning in the .lsB file given above is translated into a default invariant inserted into the loop. The line numbers and header and footer are all stripped off.

```
package body Filter_Sum is

   procedure Filter_S(D: in D_Type; R: out
   Integer)
   is
   begin
      R:= 0;
      for I in I_Type loop
```

```
--# assert true;
         if D(I) >= 0 and D(I) <= 100 then
            R:= R + D(I);
         end if;
      end loop;
   end Filter_S;

end Filter_Sum;
```

A.3 filters.adn

I called the source with numbered invariant annotations `filters.add`. The file with the default invariants added is parsed and each invariant(including any that were present in the original source) is annotated with a number.

```
package body Filter_Sum is

   procedure Filter_S(D: in D_Type; R: out
   Integer)
   is
   begin
      R:= 0;
      for I in I_Type loop
--# assert inv_number( 1 ) and true;
         if D(I) >= 0 and D(I) <= 100 then
            R:= R + D(I);
         end if;
      end loop;
   end Filter_S;

end Filter_Sum;
```

A.4 filters.prp / filters.adi

I called a sample possible output from the program analysis `filters.prp` and the final patched result `filters.adi`.

The output from the program analysis should be something of the form:

```
1:R >= 0 and R <= I * 100
```

This informtion is patched into `filters.adn` replacing the appropriate numered invariant present with the property specified in the file.

```
package body Filter_Sum is

   procedure Filter_S(D: in D_Type; R: out
   Integer)
   is
   begin
      R:= 0;
      for I in I_Type loop
--# assert R >= 0 and R <= I * 100;
         if D(I) >= 0 and D(I) <= 100 then
            R:= R + D(I);
         end if;
      end loop;
   end Filter_S;

end Filter_Sum;
```

B invade

The code of the `invade` awk script:

```
#!/bin/awk -f
#Set up stuff: Set FIELDWIDTHS so that the whitespace is preserved
BEGIN{
    mine=0;
    loop=0;
    FIELDWIDTHS="6 100";
}
#function to insert the default invariant.
    function assert(var){
        printf("--# assert %s;\n",var);
        loop=0;mine=0;
    }
#every line with a line number:
#   remove number print the rest of the line
    $1~/[0-9]+/&&$1!~/[A-Za-z\*\(\)\+\-]+/{
        for(i=2;i<=NF;i++) printf("%s ", $i);
        printf("\n");
    }
#A line with a loop on it but not end loop or a warning:
#   Set loop to 1 and if mine is also 1 insert the invariant.
    /loop/&&!/end loop/&&!/Warning/{
        loop=1;
        if(mine==1){    assert("true");}
    }
#A line with a warning on it:
#   It should be an assertion warning so set mine.
#   And if loop is already set add the invariant.
    /Warning/&&/Default assertion/{
        mine=1;
        if(loop==1){    assert("true");}
    }
#A line with end loop on it:
#   Reset loop and mine because the loop is over.
    /end loop/{
        loop = 0;
        mine = 0;
    }
END{}
```

C invasive

The code of the invasive awk script:

```
#!/bin/awk -f

#setup stuff: set the initial value for the assertion number counter
BEGIN{
    my=0;
}
#function to insert the number annotation into an invariant.
    function assert(){
        my=my+1;
        printf("inv_number( %d ) and ",my);
    }
#On finding an assertion in the code add a number annotation to it.
#   And print the rest of the line.
    $1~"--#"{
        printf("--# assert ");
        assert();
```

```
        for(i=3;i<=NF;i++) printf("%s ", $i);
        printf("\n");
    }
#Anything other than an assertion: Print the line as it is.
    $1!~"--#"{print $0}
END{}
```

D invader

The code of the invader shell script:

```
#!/bin/bash

#Find the directory containing the script
SCRIPTVAL=`which $0`
SCRIPTPATH=`echo $SCRIPTVAL | sed -e "s/[^/]*$//"`

#Setup stuff: get values for N and P from the property file.
Enn=`awk -F: '{print $1}' $2`
Pee=`awk -F: '{print $2}' $2`

#Pass N and P to the invaderiser awk script.
awk -f $SCRIPTPATH/invaderiser N=$Enn P="$Pee" $1
```

The code of the invaderiser awk script:

```
#Set up stuff: Set default values for N and P.
#   Values for both N and P should be provided on the command line,
#   to override these defaults.
BEGIN{
    N=0;
    P="true";
}
#An assertion in the code:  Check if it's number annotation matches
#   the value given for N and if so replace it with the assertion of P.
    $1~"--#"{
        num = $4;
        if(num==N)
            printf("--# assert %s;\n",P);
        else
            print $0;
    }
#Anything other than an assertion: Print the line as it is.
    $1!~"--#"{print $0}
END{}
```