A Comparison of the Program Provers FPP, NPPV and SPARK

Jürgen F H Winkler

Friedrich Schiller University

winkler@informatik.uni-jena.de

Course "Mechanical Program Verification"

ELTE, Budapest Oktober 2007

OVERVIEW

What is and what does a program prover?

FPP (= Frege Program Prover)

NPPV (= New Paltz Program Verifier)

SPARK (SPADE Ada Real-time Kernel)

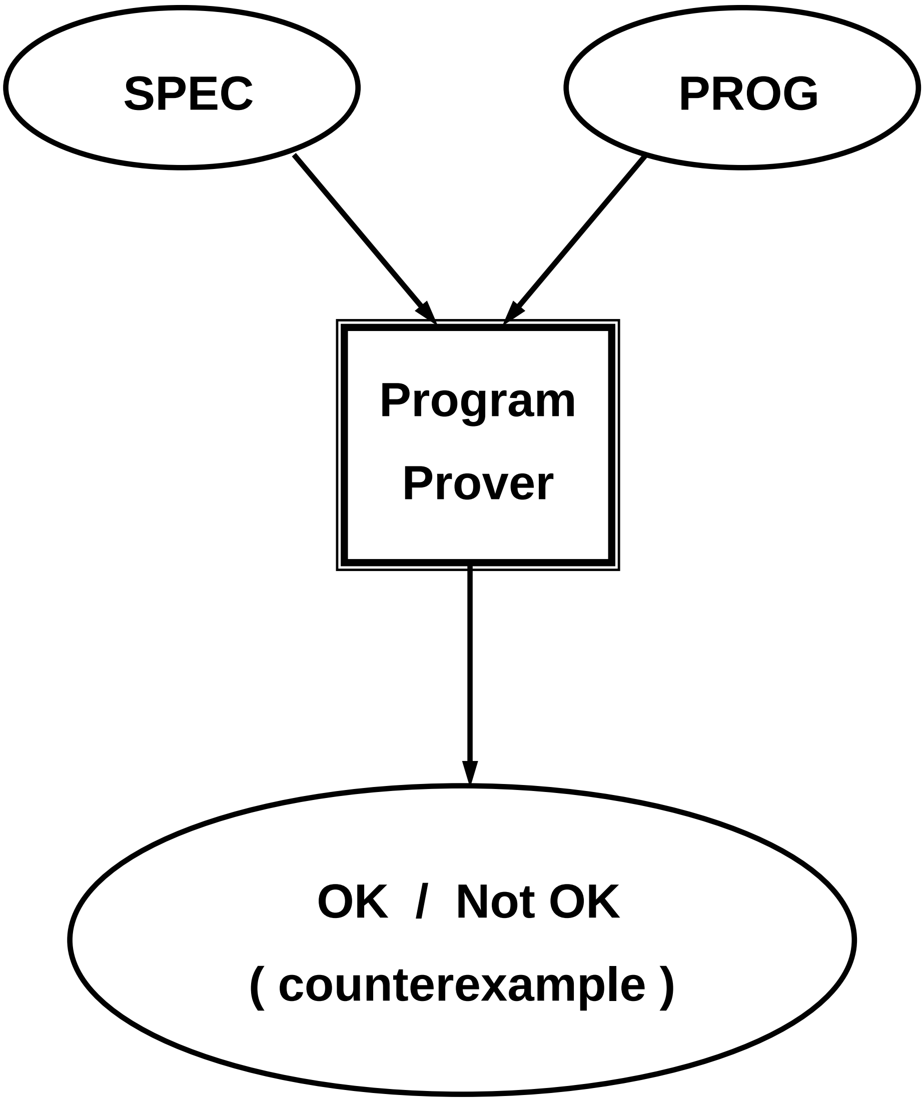
Comparison Results

Some examples

Outlook

References

WHAT IS A PROGRAM PROVER ?



SPEC: SPECIFIES WHAT THE PROGRAM P SHALL DO

Example1

increment the value of X

new / final value of X = old / initial value of X + 1

X' = X +1

e.g. Hehner, Z

pre: X = Xk

post: X = Xk+1

suitable for practical program development

Example 1 is too naïve

In real programs variables have a ***finite range***

i.e. at all points where the variable X can be observed $X \in Type(X)$ must hold: e.g. $0 \leq X \leq 100$

pre: $X = Xk \wedge 0 \leq X \leq 100$
post: $X = Xk+1 \wedge 0 \leq X \leq 100$

Xk is a constant and NOT a program variable

(also called specification variable)

THE PROGRAM P IS INSERTED BETWEEN PRE AND POST

pre: $X = Xk \wedge 0 \leq X \leq 100$

P

post: $X = Xk+1 \wedge 0 \leq X \leq 100$

Often we find the naive Increment program X := X + 1;

pre: $X = Xk \wedge 0 \leq X \leq 100$

X := X + 1;

post: $X = Xk+1 \wedge 0 \leq X \leq 100$

Let Xk=100

pre: $X = 100 \wedge 0 \leq X \leq 100$

X := X + 1;

post: $X = 100+1 \wedge 0 \leq X \leq 100$ <=== !!!

===> P is NOT correct

WHAT IS PROGRAM CORRECTNESS ?

Total correctness:	**Start P in a state in which pre is true** **P terminates after finite time and then post is true**

Partial correctness:	**Start P in a state in which pre is true** **After regular termination of P post is true**

HOW TO CHECK THIS MECHANICALLY ?

VERIFICATION CONDITIONS (VCs)

The semantics of a language element E defines which pairs (pre, post) are consistent with E i.e.

pre

E

post

is correct

This relation between pre, post and E is called a *verification condition* (VC).

***Weakest precondition* (wp) and *strongest postcondition* (sp) are two calculi for VCs.**

All three tools use wp.

VC for wp : $(\forall \text{ Var} : \ \text{pre} \Rightarrow \text{wp(E, post)})$

Example : wp for assignment

$\text{wp}(\text{''X := e ;''}, \text{post}) \equiv e \in \text{Type(X)} \land \text{post}^X_e$

This can be computed mechanically

WHAT DOES AN AUTOMATIC PROGRAM PROVER ?

--!pre: $X = Xk \wedge 0 \le X \le 100$
X := X+1;
--!post: $X = Xk+1 \wedge 0 \le X \le 100$

SPEC + PROG
(legal Ada)

(FPP)

1) Generate VCs

2) Try to prove VCs

3) If no success
generate FCs

FC: Falsification
Condition

--!pre: $X = Xk \wedge 0 \le X \le 100$
--!wp: $X+1 = Xk+1 \wedge 0 \le X+1 \le 100$
--!VC: $X = Xk \wedge 0 \le X \le 100 \Rightarrow X = Xk \wedge -1 \le X \le 99$
--! not proved
--! FC: X=100
X := X+1;
--!post: $X = Xk+1 \wedge 0 \le X \le 100$

FPP - 1

Programming Language

	Subset of Ada:	**integer and Boolean**
		null, assg, IF, CASE, WHILE, FOR
Assertion Lang:		**Ada Boolean expressions**
		+ quantifiers (forall, exists)
		+ implication (=>)
		+ predefined functions (min, sum, ...)
		--!pre: x >= 0;
Functionality		**compute wp**
		compute and prove VC
		compute FC

Application in the WWW

Input :	**Spec + Prog in one “file“ (legal Ada)**
Output:	**Spec + Prog + Result in one “file“**

Authors:	**Knappe/Kauer/Winkler** **Friedrich Schiller Univ Jena** **http://psc.informatik.uni-jena.de/FPP/FPP-main.htm**
Intention:	**experimental system for educational purposes**

N P P V

Programming Language	
Subset of Pascal	**integer and integer array**
	assg, IF, WHILE, FOR
Assertion Language	**Pascal Boolean expressions**
	{ x >= 0 }
Functionality	**compute and prove VC**
Standalone application	
Input	**Spec + Prog in one “file“** **may be illegal Pascal**
Output	**Result in one “file“**
Author:	**P. Gumm** **State Univ of New York at New Paltz** **(now: Univ of Marburg)**

http://www.mathematik.uni-marburg.de/~gumm/NPPV/nppv.html

N P P V

```
  { x = xk and 0<=x and x<=100 }
BEGIN     x:=x+1    END
{ x=xk+1 and 0<=x and x<=100 }
```

SPEC + PROG
(legal Pascal)

(NPPV)

1) Generate VCs

2) Try to prove VCs

```
Trying to prove : work.ver
Generating verification conditions...      O.K.
=======================================
=== Verification Condition No.: 1 ===
x=xk AND 0<=x AND x<=100
   ==>     x+1=xk+1 AND 0<=x+1 AND x+1<=100
--------- Remains to prove  ---------
0<=xk AND xk<=100
         ==>
                0<=xk+1 AND xk+1<=100
=======================================
     FINISHED
```

S P A R K

Programming Language	
Subset of Ada	**quite large subset**
Assertion Language	**Ada Boolean expressions** **+ quantifiers** **+ implication** **--# assert x >= 0 ;**
Functionality	**compute and prove VCs**
Standalone application	
Input	**Spec + Prog in one file** **legal Ada**
Output	**after Examiner: 6 output files** **(.fdl, .lst, .out, .rep, .rls, .vcg)** **after Simplifier: 2 additional outp. files** **(.slg, .siv)**
Supplier	**Praxis Critical Systems Ltd.** **System from CD in [Bar 2000]**

SPARK

```
--# assert X=Xk and 0<=x and x<=100;
X := X+1;
--# assert X=Xk+1 and 0<=x and x<=100;
```

SPEC + PROG
(legal Ada)
(only fragment of input)

(SPARK)

1) Generate VCs (Examiner)

2) Try to prove VCs (Simplifier)

For path(s) from assertion of line 10 to assertion of line 12:

```
procedure_ex_a15vc_2.
H1:    0 <= xk .
H2:    xk <= 100 .
       ->
C1:    xk <= 99 .
```

$H1 \wedge H2 \Rightarrow C1$

Comparison between FPP, NPPV and SPARK

FPP is an experimental program prover

supports only some language elements

contains errors

- ⇨ **we wanted to know where we stand**

first comparison FPP vs NPPV in 1999

(Kauer / Winkler [KW 99])

second comparison FPP, NPPV, SPARK in 2002

(Freining / Kauer / Winkler [FKW 2002])

How to compare ?

- ⇨ **functional and usability aspects**
- ⇨ **26 examples: 20 from the NPPV distribution**
  5 from Kauer
  1 from Gravell and Hehner [GH 99]

Property	FPP	NPPV	SPARK-aut
programming language	subset of Ada; FRAGMENTS	subset of Pascal FRAGMENTS	subset of Ada; COMPLETE PROGRAMS
assertion language	subset of Ada expressions extended with *quantifiers*, *implication* and the additional functions abs, min, max, ggt, sum, factorial, fib	subset of Pascal expressions, enclosed in { }; true is expressed by {}	subset of Ada expressions extended with *quantifiers*, *implication*, equivalence, proof variables and functions, and update of structured objects
form of assertions	special comments: --!	{}comments and [...]	special comments: --#
multiline assertions	supported	supported	supported
supported types	integer and Boolean	integer, array with integer index type and integer component type	all, except tagged, access, task, and exception
supported statements	NULL, assignment, IF, CASE, FOR and WHILE loop	assignment, IF, FOR- and WHILE loop	all, except goto and tasking
proof of loops	precondition, postcondition, invariant and for WHILE loops a termination function has to be supplied	only invariant required, termination function for WHILE loops optional	invariant can be inserted in body, termination has to be expressed by assertions
output	in a file that has the same name as the input file, but a different extension; output contains the statements, the VCs and the result together	optional in a file: session.log; output contains only verification conditions and results	in up to 8 different files
usage	local or via WWW	local	local
pretty printing	supported	not supported	not supported
simplification of expr.	performed to a certain extent	not performed	performed to a certain extent
explicit comp. of wp	possible	not possible	not possible
theorem proving	possible e.g. with null statement	possible e.g. with x := x	possible e.g. with null statement
implementation language	Ada, C and Mathematica	Visual Prolog	SPARK, Prolog
proving power	higher than NPPV and SPARK-aut	only trivial	rather limited
automatic theorem prover	mexana, an extension of Analytica	simple rewrite system	automatic Simplifier

Merkmale von FPP, NPPV und SPARK-aut

No.	Example	Remark	Source	FPP			NPPV			SPARK-aut		
				proved	#VC	#OK	proved	#VC	#OK	proved	#VC	#OK
1	abs	no abs function in NPPV	Kauer	PROV	1	1	NOP	2	1	NOP	2	0
2	array	no arrays in FPP	Gumm	n.a.			SOLV	1	1	NOS	1	0
3	assrek	no termination function	Gumm	NOS	0	0	SOLV	4	4	SOLV	4	4
4	factfor		Gumm	PROV	5	5	NOP	5	2	NOP	4	1
5	factforty		Gumm / Kauer	PROV	5	5	NOP	5	1	NOP	4	1
6	fastmul	no termination function	Gumm	NOP	6	5	NOP	6	4	NOP	9	7
7	fastmult		Gumm	NOP	10	7	NOP	14	7	NOP	9	7
8	fastmultty	too many clauses (FPP)	Gumm / Kauer	NOP	0	0	NOP	14	7	NOP	10	7
9	fibo	no termination function	Gumm	PROV	3	3	NOP	4	2	NOP	4	2
10	fibot		Gumm	PROV	6	6	NOP	6	3	NOP	5	3
11	fibotty		Gumm / Kauer	NOP	6	5	NOP	6	3	NOP	5	2
12	gauss	no termination function	Gumm	PROV	4	4	NOP	4	3	NOP	4	3
13	gausst		Gumm	PROV	6	6	NOP	6	5	NOP	5	4
14	gaussty		Gumm / Kauer	PROV	6	6	NOP	6	4	NOP	5	3
15	linrek	no termination function	Gumm	SOLV	7	7	SOLV	7	7	SOLV	8	8
16	linsearch	no quantifiers in NPPV	Kauer	PROV	5	5	n.a.			NOP	4	1
17	nested_for		Kauer	PROV	9	9	NOP	8	5	NOP	9	7
18	proof		Gumm	SOLV	4	4	SOLV	4	4	SOLV	4	4
19	quad		Kauer	PROV	5	5	NOP	5	4	NOP	5	4
20	root		Kauer	PROV	5	5	NOP	5	4	NOP	4	3
21	swap1		Gumm	PROV	1	1	PROV	1	1	PROV	1	1
22	swap2	infinite ranges	Gumm	PROV	1	1	PROV	1	1	PROV	1	1
23	swap2ty	fin ranges, prog incorrect	Gumm	NOP	3	0	NOP	3	0	NOP	3	0
24	swap2ty2		Gumm / Kauer	PROV	3	3	NOP	3	1	PROV	3	3

No.	Example	Remark	Source	FPP			NPPV			SPARK-aut		
				proved	#VC	#OK	proved	#VC	#OK	proved	#VC	#OK
25	swap3		Gumm	SOLV	1	1	SOLV	1	1	SOLV	1	1
26	cube		[GH 99]	PROV	5	5	NOP	5	3	NOP	5	3
	Summary			19 (25)	107	99	7 (25)	126	78	7 (26)	119	80
	Summary			**76 %**	**93 %**		**28 %**	**62 %**		**27 %**	**67 %**	

Tabelle 6.2. Ergebnisse für die 26 Beispiele

.

EXAMPLE-21

21. swap1

Swapping the values of two variables using an auxiliary variable.

Input to NPPV

```
{ x = A and y = B }
BEGIN
  temp := x ;
  x    := y ;
  y    := temp
END
{ x = B and y = A }
```

Output from NPPV

```
=====================================
=== Verification Condition No.: 1 ===

x=A AND y=B
    ==>
        y=B AND x=A
=========  Proof succeeded  =========
=====================================
```

21. swap1

Swapping the values of two variables using an auxiliary variable.

Input to FPP

```
    --  Example 21
    --!pre: x = x_i and y = y_i;
 temp := x ;
 x    := y ;
 y    := temp;
    --!post: x = y_i and y = x_i;
```

Output from FPP

```
FPP (Frege Program Prover) University of Jena, Germany
User: 141.35.14.241       At: 1999.09.24, 10:24
The answer to your query is:

--!pre   : (x = x_i AND y = y_i)
--> wp   : (y = y_i AND x = x_i)
--> vc   : (x = x_i AND y = y_i ==> y = y_i AND x = x_i)
--> Result: proved
temp := x;
x := y;
y := temp;
--!post      : (x = y_i AND y = x_i)
```

21. swap1

Swapping the values of two variables using an auxiliary variable.

Input to SPARK

```
  -- ex_21vc   Examiner: verification = vc

   --# main_program;
procedure ex_21vc (X: in out integer; Y: in out integer)
   --# derives X, Y from X, Y;
is
   temp: integer := 0;
   x_i : integer := 0;
   y_i : integer := 0;
begin
   x_i := x;
   y_i := y;

   --# assert x = x_i and y = y_i;
   temp := x ;
   x    := y ;
   y    := temp;
   --# assert x = y_i and y = x_i;

end ex_21vc;
```

Output from SPARK examiner

Ex_21vc.lst

```
          ******************************************************
                          Listing of SPARK Text
  SPARK95 Examiner with VC and RTC Generator Release 5.01 / 08.00
                          Demonstration Version
          ******************************************************


                      DATE : 02-JAN-2002 09:40:21.22

 Line
    1    -- ex_21vc   Examiner: verification = vc
    2
    3     --# main_program;
    4  procedure ex_21vc (X: in out integer; Y: in out integer)
    5     --# derives X, Y from X, Y;
    6  is
    7
    8     temp: integer := 0;

 !!! (  1)  Flow Error        : 54: The initialization at declaration
of temp is  ineffective.

    9     x_i : integer := 0;

 !!! (  2)  Flow Error        : 54: The initialization at declaration
of x_i is   ineffective.

   10     y_i : integer := 0;

 !!! (  3)  Flow Error        : 54: The initialization at declaration
of y_i is   ineffective.

   11
   12  begin
   13     x_i := x;
          ^4
 !!! (  4)  Flow Error        : 10: Ineffective statement.

   14     y_i := y;
          ^5
```

```
 !!! (  5)  Flow Error        : 10: Ineffective statement.


       --# assert x = x_i and y = y_i;
       temp := x ;
       x    := y ;
       y    := temp;
       --# assert x = y_i and y = x_i;

    end ex_21vc;

 !!! (  6)  Flow Error        : 33: The variable x_i is neither ref-
erenced nor
            exported.
 !!! (  7)  Flow Error        : 33: The variable y_i is neither ref-
erenced nor exported.
 !!! (  8)  Flow Error        : 50: The imported value of X is not
used in the  derivation of X.
 !!! (  9)  Flow Error        : 50: The imported value of Y is not
used in the derivation of Y.

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

Ex_21vc.siv

```
*******************************************************
                Semantic Analysis of SPARK Text
SPARK95 Examiner with VC and RTC Generator Release 5.01 / 08.00
                            Demonstration Version

*******************************************************

 CREATED   02-JAN-2002,   09:40:21     SIMPLIFIED   02-JAN-2002,
09:40:29
          (Simplified by SPADE Simplifier, Version 1.4)

                              procedure ex_21vc

 For path(s) from start to assertion of line 16:

 procedure_ex_21vc_1.
 *** true .          /* all conclusions proved */


 For path(s) from assertion of line 16 to assertion of line 20:

 procedure_ex_21vc_2.
 *** true .          /* all conclusions proved */


 For path(s) from assertion of line 20 to finish:

 procedure_ex_21vc_3.
 *** true .          /* all conclusions proved */
```

EXAMPLE-22

22. swap2 Tricky but unsafe version of swapping the values of two variables without an auxiliary variable.

Input to NPPV

```
{ x = M and y = N }
BEGIN
  x := x - y;
  y := x + y;
  x := y - x
END
{ x = N and y = M }
```

Output from NPPV

```
=====================================
=== Verification Condition No.: 1 ===

x=M AND y=N
   ==>
      x-y+y-(x-y)=N AND x-y+y=M
=========  Proof succeeded  =========
=====================================
```

22. swap2 Tricky but unsafe version of swapping the values of two variables without an auxiliary variable.

Input to FPP

```
    --  Example 22
    --!pre : x = x_i and y = y_i;
 x := x - y;
 y := x + y;
 x := y - x;
    --!post: x = y_i and y = x_i;
```

Output from FPP

```
FPP (Frege Program Prover) University of Jena, Germany
User: 141.35.14.241       At: 1999.09.24, 10:31
The answer to your query is:

--!pre       : (x = x_i AND y = y_i)
--> wp       : (y = y_i AND x = x_i)
--> vc       : (x = x_i AND y = y_i ==> y = y_i AND x = x_i)
--> Result: proved
x := x - y;
y := x + y;
x := y - x;
--!post      : (x = y_i AND y = x_i)
```

Since neither NPPV nor FPP take the limited ranges of integer types into account both say that the programs are correct.
In typical implementations of Pascal and Ada the programs are not correct, because the difference in ”x := x-y;“ cannot be computed for all legal combinations of x and y [Win 90].

22. swap2 Tricky but unsafe version of swapping the values of two variables without an auxiliary variable.

Input to SPARK

Ex_22vc.ada

```
  -- ex_22vc   Examiner: verification = vc

   --# main_program;
procedure ex_22vc (X: in out integer; Y: in out integer)
   --# derives X, Y from X, Y;
is
   x_i: integer := 0;
   y_i: integer := 0;
begin
   x_i := x;
   y_i := y;

   --# assert x = x_i and y = y_i;
   x := x - y;
   y := x + y;
   x := y - x;
   --# assert x = y_i and y = x_i;

end ex_22vc;
```

Output from SPARK

Ex_22vc.siv

```
**********************************************************
              Semantic Analysis of SPARK Text
SPARK95 Examiner with VC and RTC Generator Release 5.01 /
08.00
                             Demonstration Version

**********************************************************

 CREATED 02-JAN-2002, 09:44:01  SIMPLIFIED 02-JAN-2002,
09:44:09
  (Simplified by SPADE Simplifier, Version 1.4)

                               procedure ex_22vc

 For path(s) from start to assertion of line 15:

 procedure_ex_22vc_1.
 *** true .          /* all conclusions proved */


For path(s) from assertion of line 15 to assertion of line
19:

 procedure_ex_22vc_2.
 *** true .          /* all conclusions proved */

 For path(s) from assertion of line 19 to finish:

 procedure_ex_22vc_3.
 *** true .          /* all conclusions proved */
```

EXAMPLE-23

23. swap2ty The same as example 22 but with type checking assertions.

Input to NPPV

```
{ x=M and y=N and -100<=x and x <= 100 and -100 <= y and y <= 100 }
  BEGIN
    x := x - y;
  {  -100 <= x and x <= 100 and -100 <= y and y <= 100 }
    y := x + y;
  {  -100 <= x and x <= 100 and -100 <= y and y <= 100 }
    x := y - x
  END
{ x=N and y=M and -100<=x and x <= 100 and -100 <= y and y <= 100 }
```

Output from NPPV

```
=====================================
=== Verification Condition No.: 1 ===

x=M AND y=N AND -100<=x AND x<=100 AND -100<=y AND y<
=100   ==>
      -100<=x-y AND x-y<=100 AND -100<=y AND y<=100

--------- Remains to prove  ---------
-100<=M AND M<=100 AND -100<=N AND N<=100   ==>
                -100+N<=M AND M<=100+N
=====================================

=====================================
=== Verification Condition No.: 2 ===

-100<=x AND x<=100 AND -100<=y AND y<=100   ==>
       -100<=x AND x<=100 AND -100<=x+y AND x+y<=100

--------- Remains to prove  ---------
-100<=x AND x<=100 AND -100<=y AND y<=100 ==>
                -100<=x+y AND x+y<=100
=====================================
```

```
=====================================
=== Verification Condition No.: 3 ===

-100<=x AND x<=100 AND -100<=y AND y<=100
         ==>
                y-x=N AND y=M AND -100<=y-x AND y-x<=
100 AND -100<=y AND y<=100

--------- Remains to prove  ---------
-100<=x AND x<=100 AND -100<=y AND y<=100
         ==>
                y=M AND y-x=N AND -100+x<=y AND y<=10
0+x
=====================================
```

23. swap2ty

The same as example 22 but with type checking assertions.

Input to FPP

```
    --  Example 23
    --!pre : x=x_i and -100 <= x and x <= +100 and
    --!pre : y=y_i and -100 <= y and y <= +100;
 x := x - y;
    --!post: -100 <= x and x <= +100 and -100 <= y and y <= +100;
 y := x + y;
    --!post: -100 <= x and x <= +100 and -100 <= y and y <= +100;
 x := y - x;
    --!post: x=y_i and -100 <= x and x <= +100 and
    --!post: y=x_i and -100 <= y and y <= +100;
```

Output from FPP

```
FPP (Frege Program Prover) University of Jena, Germany
User: 141.35.14.241       At: 1999.09.24, 12:28
The answer to your query is:

--!pre :     (x = x_i) AND (-100 <= x) AND (x <= 100)
-->           AND (y = y_i) AND (-100 <= y) AND (y <= 100)
--> wp : (-100 <= x - y AND x - y <= 100 AND -100 <= y AND y <= 100)
--> vc :      (x = x_i) AND (-100 <= x) AND (x <= 100)
-->       AND y <= 100) AND (y = y_i) AND (-100 <= y) AND (y <= 100)
-->  ==> (-100 <= x - y AND x - y <= 100 AND -100 <= y AND y <= 100)
--> Result: not proved
--> fc  :  (-100+x_i-y >= 1) AND (100-x_i >= 0) AND (100+x_i >= 0)
-->     AND (100 + y >= 0) AND (100 - y >= 0)

x := x - y;
--!post : (-100 <= x AND x <= 100 AND -100 <= y AND y <= 100)
--> wp  : (-100 <= x AND x <= 100 AND -100 <= x + y AND x + y <=
100)
--> vc  : (-100 <= x AND x <= 100 AND -100 <= y AND y <= 100)
-->   ==> (-100 <= x AND x <= 100 AND -100 <= x+y AND x+y <= 100)
--> Result: not proved
--> fc  : (-100 - x - y >= 1) AND (100 + x >= 0) AND (100 - y >= 0)
-->   AND (100 + y >= 0) AND (100 - x >= 0)

y := x + y;
```

```
--!post : (-100 <= x AND x <= 100 AND -100 <= y AND y <= 100)
--> wp  : (-x + y = y_i) AND (-100 <= -x + y) AND (-x + y <= 100)
-->   AND (y = x_i) AND (-100 <= y) AND (y <= 100)
--> vc  : (-100 <= x AND x <= 100 AND -100 <= y AND y <= 100)
-->  ==>  (-x + y = y_i) AND (-100 <= -x + y) AND (-x + y <= 100)
-->      AND (y = x_i) AND (-100 <= y) AND (y <= 100)
--> Result: not proved
--> fc  : (100 - x >= 0) AND (100 + x >= 0) AND (100 + y >= 0)
-->   AND (y /= x_i) AND (100 - y >= 0)

x := y - x;
--!post : (x = y_i) AND (-100 <= x) AND (x <= 100)
-->   AND (y = x_i) AND (-100 <= y) AND (y <= 100)
```

When the limited domains of integer types are checked both provers cannot prove the program. The program is not correct, i.e. the falsification conditions are true. The first falsification condition in the output from FPP is equivalent to

$$-100 \leq x \leq +100 \;\wedge\; -100 \leq y \leq +100 \;\wedge\; x - y \geq 101$$

which is satisfied by e.g. $x = 100 \;\wedge .\; y = -1$ which is a legal pair of values for x and y.

23. swap2ty The same as example 22 but with type checking assertions.

Input to SPARK

Ex_23vc.ada

```
  -- ex_23vc   Examiner: verification = vc

   --# main_program;
procedure ex_23vc (X: in out integer; Y: in out integer)
   --# derives X, Y from X, Y;
is
   x_i : integer := 0;
   y_i : integer := 0;
begin
   x_i := x;
   y_i := y;

   --# assert x = x_i and -100 <= x and x <= +100 and
   --#        y = y_i and -100 <= y and y <= +100;
   x := x - y;

--# assert  -100<=x and x<=+100 and -100<=y and y<=+100;
   y := x + y;

--# assert  -100<=x and x<=+100 and -100<=y and y<=+100;
   x := y - x;

   --# assert  x = y_i and -100 <= x and x <= +100 and
   --#         y = x_i and -100 <= y and y <= +100;

end ex_23vc;
```

Output from SPARK

Ex_23vc.siv

```
*******************************************************
        Semantic Analysis of SPARK Text
SPARK95 Examiner with VC and RTC Generator Release
5.01/08.00
               Demonstration Version
 *******************************************************
 CREATED  02-JAN-2002,  09:45:55   SIMPLIFIED  02-JAN-2002,
09:46:03
   (Simplified by SPADE Simplifier, Version 1.4)

                              procedure ex_23vc

 For path(s) from start to assertion of line 16:

 procedure_ex_23vc_1.
 H1:    x >= integer__first .
 H2:    x <= integer__last .
 H3:    y >= integer__first .
 H4:    y <= integer__last .
        ->
 C1:    - 100 <= x .
 C2:    x <= 100 .
 C3:    - 100 <= y .
 C4:    y <= 100 .
```

```
 For path(s) from assertion of line 16 to assertion of line
20:

 procedure_ex_23vc_2.
 H1:    - 100 <= x_i .
 H2:    x_i <= 100 .
```

```
H3:     - 100 <= y_i .
H4:     y_i <= 100 .
        ->                          false
C1:     - 100 <= x_i - y_i .
C2:     x_i - y_i <= 100 .

For path(s) from assert. of line 20 to assert. of line 23:

procedure_ex_23vc_3.
H1:     - 100 <= x .
H2:     x <= 100 .
H3:     - 100 <= y .
H4:     y <= 100 .                    false
        ->
C1:     - 100 <= x + y .
C2:     x + y <= 100 .

For path(s) from assert. of line 23 to assert. of line 26:

procedure_ex_23vc_4.
H1:     - 100 <= x .
H2:     x <= 100 .
H3:     - 100 <= y .
H4:     y <= 100 .                     false
        ->
C1:     y - x = y_i .
C2:     - 100 <= y - x .
C3:     y - x <= 100 .
C4:     y = x_i .
```

```
For path(s) from assertion of line 26 to finish:

procedure_ex_23vc_5.
*** true .          /* all conclusions proved */
```

OUTLOOK: FPP-2

Programming Language

	Subset of Ada:	**integer and Boolean**
		user defined types
		array, record, procedure
		null, assg, IF, CASE, WHILE, FOR

Assertion Lang:	**Ada Boolean expressions**
	+ quantifiers (forall, exists)
	+ implication (=>)
	+ predefined functions (min, sum, ...)
	--!pre: x >= 0;
Functionality	**compute wp** ***with range checks***
	compute and prove VC
	compute FC

Application in the WWW

Input :	**Spec + Prog in one “file“ (legal Ada)**
Output:	**Spec + Prog + Result in one “file“**

Outlook: MPV in Practice

Often people working in program verification complain that PV is no used in the SW industry

What are reasons for this deplorable situation?

- **current results in PV too theoretical**
- **SW engineers do not have the suitable training / education**
- **no tools available**
- **education in school and university does not contain enough logic and discrete mathematics**

  (logic courses at the university mainly are mostly about logic systems but do not lead to fluency in manipulation of logic formulas as compared with the skills students have in the manipulation of arithmetic formulas)

Experiment to support this last point:

$x+y>x-y \lor y<2x \Rightarrow$ true

x, y $\in \mathbb{Z}$ (let us (for a moment) be in Cantor's Paradise)

If you look at:

$$(x+2y+x*y)*0$$

it is much more easier (currently for most people)

$a \Rightarrow true \;\equiv\; true$

$a*0 = 0$

$(a \vee b) \Rightarrow c$ **$(a + b) * c$**

$c \Rightarrow (a \vee b)$ **$c * (a + b)$**

Reason: there is much more training in arithmetic formula manipulation than in logic formula manipulation until the end of university education (diploma/master)

Arithmetic: ca 14 years

Logic: ca 1-2 years

(Figures for Germany)

REFERENCES

FPP

Freining, Carsten; Kauer, Stefan; Winkler, Jürgen F. H.
Ein Vergleich der Programmbeweiser FPP, NPPV und SPARK.
Ada-Deutschland Tagung 2002. Shaker Verlag, Aachen, 2002.
S. 127..145

http://psc.informatik.uni-jena.de/FPP/FPP-main.htm

NPPV

Gumm, Heinz-Peter; Sommer, Manfred: Einführung in die Informatik. R. Oldenbourg Verlag. München 2002.
3-486-25635-1

http://www.mathematik.uni-marburg.de/~gumm/NPPV/nppv.html
(visited 2001.Sep.04)

SPARK

Barnes, John: High Integrity Ada - The SPARK Approach -.
Addison-Wesley, Harlow etc., 2000. 0-201-17517-7

REST

A COMPARISON OF THE PROGRAM PROVERS FPP, NPPV AND SPARK

Jürgen F H Winkler

Friedrich Schiller University

winkler@informatik.uni-jena.de

Course “Mechanical Program Verification”

ELTE, Budapest September 2006