Mechanical Program Verification – Part 2

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany

ELTE, Budapest, 8 - 12 Oct 2007

Overview

- Historical Overview, Basic Concepts, Realistic Progr Verific
- Mechanical Program Verification (MPV)
- Comparison of 3 Automatic Program Provers (APP)
- The Frege Program Prover (FPP) in More Detail
- Mechanical Generation of Invariants for FOR-Loops
- Problems of FPP (and others)
- Towards Realistic Verification Conditions (VC)
- Summary

Part 2

- *Mechanical* Program Verification (MPV)
- Tools: FPP, NPPV, SPARK

My real topic / concern is

Mechanical Verification of ***Real Programs***

=> before we can do verification (i.e. build an APP)

we need the VCs for real programs

i.e. we have to understand
how real programs really work: e.g. v := v;

Mechanical Program Verification (MPV)

Verification : $p_o \leq$ (pre, post) is the really important thing

$\leq$: conformance relation

$p_o \leq$ (pre, post) : conformance condition (CC)
verification condition (VC)

Mechanical : compute the VC ***mechanically***
and try to prove it ***mechanically***

- by hand (tedious and error-prone $\Rightarrow$ unfeasible)
- using a tool: automatic program prover (APP)

=> MPV = APP computes and tries to prove VC

Mechanical Program Verification: Example

Computation of the mean of two numbers:

mathematically: $mean(a,b) = (a +_m b)/_m 2$ especially: $mean(a, a) = a$

In 2006 the binary search in the Java class library worked incorrectly, because it used the naive formula (a+b)/2 (Joshua Bloch) why ???
(http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html)

How to compute it in a finite domain [min, max] $(min \le max)$

Q: $\langle \forall a, b \in [min, max]: min \le (a +_m b)/_m 2 \le max \rangle$ => mean is suited for fin dom

a/2 + b/2 ????

Not very good, even in float: succ(0.0)/2.0 + succ(0.0)/2.0) = ???

succ(0.0)/2.0 + succ(0.0)/2.0) = 0.0 !!!! (IEC 60559, IEEE 754)

Mechanical Program Verification: Example

Better formula (improvement of Kahan's formula by Jürgen Winkler*)

```
mean(a, b)  =  if  sig(a)=sig(b)
               then   if abs(a)<abs(b)
                      then  a+(b-a)/2;
                      else  b+(a-b)/2;
                      end if;
               else   (a+b)/2;
               end if;
```

everything OK ???

Unfortunately not: use of abs( ) $\Rightarrow$ domain: max $\geq$ |min|

*): Kahan, W.: Analysis and Refutation of the LCAS. SIGPLAN Notices 27,1 (1992) 61..74
Kahan and Winkler were only interested in the domains of IEEE 754 / IEC 60559

```
--!pre: a=a_i and b=b_i and -100<=a and a<=100 and  -100<=b and b<=100;
if sig(a)=sig(b) then
   if abs(a)<abs(b) then
      --!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100 and
      --!pre: sig(a)=sig(b) and abs(a)<abs(b) and -100<=(b-a) and (b-a)<=100 and
      --!pre: -100<=a+(b-a)/2 and a+(b-a)/2<=100;
      m:=a+(b-a)/2;
      --!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
   else
      --!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100 and
      --!pre: sig(a)=sig(b) and abs(a)>=abs(b) and -100<=(a-b) and (a-b)<=100
      --!pre: and -100<=b+(a-b)/2 and b+(a-b)/2<=100;
      m:=b+(a-b)/2;
      --!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
   end if;
else
   . . .
end if;

--!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
```

```
--!pre: a=a_i and b=b_i and -100<=a and a<=100 and  -100<=b and b<=100;
if sig(a)=sig(b) then
   if abs(a)<abs(b) then
      --!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100 and
      --!pre: sig(a)=sig(b) and abs(a)<abs(b) and -100<=(b-a) and (b-a)<=100 and
      --!pre: -100<=a+(b-a)/2 and a+(b-a)/2<=100;
      m:=a+(b-a)/2;
      --!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
   else
      --!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100 and
      --!pre: sig(a)=sig(b) and abs(a)>=abs(b) and -100<=(a-b) and (a-b)<=100
      --!pre: and -100<=b+(a-b)/2 and b+(a-b)/2<=100;
      m:=b+(a-b)/2;
      --!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
   end if;
else
   . . .
end if;

--!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
```

One VC (proof obligation) is

```
--!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100 and
--!pre: sig(a)=sig(b) and abs(a)<abs(b) and -100<=(b-a) and (b-a)<=100 and
--!pre: -100<=a+(b-a)/2 and a+(b-a)/2<=100;
m:=a+(b-a)/2;
--!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;
```

VC $\equiv \langle \forall$ vars: pre $\Rightarrow$ wp("m:=a+(b-a)/2;", post)$\rangle$

Usually, a great number of such VCs: VC1, …, VCn

theoretically one big conjunction: VC1 $\wedge$ … $\wedge$ VCn

but it is easier to prove smaller VCs (Turing)

Proof by FPP:

```
    --> vc      :         (a = a_i)  AND (b = b_i)  AND (-100 <= a)   AND (a <= 100)
-->                   AND (-100 <= b)  AND (b <= 100)
-->                   AND (sig(a) = sig(b))
-->                   AND (Abs(b) >= 1 + Abs(a))
-->                   AND (-100 <= -a + b)   AND (-a + b <= 100)
-->                   AND (-100 <= a + (-a + b)/2)  AND (a + (-a + b)/2 <= 100)
-->               ==>     (a = a_i)
-->                   AND (b = b_i)
-->                   AND (a + (-a + b)/2 = (a + b)/2)
-->                   AND (-100 <= a + (-a + b)/2)   AND (a + (-a + b)/2 <= 100)
    --> Result: proved
    m := a + (b - a) / 2;
    --!post     : (a = a_i AND b = b_i AND m = (a + b)/2 AND -100 <= m AND m <= 100)
```

But FPP cannot prove the whole :

--!pre: a=a_i and b=b_i and -100<=a and a<=100 and -100<=b and b<=100;

--> vc : (a = a_i) . . .

--> pretty printed formula too long

--> Result: to many clauses generated; not proved

```
if sig(a)=sig(b) then
  if abs(a)<abs(b) then
    { pre2 }  m:=a+(b-a)/2;  { post2 }
  else
    { pre3 }  m:=b+(a-b)/2;  { post3 }
  end if;
else
  { pre4 }   m:=(a+b)/2;  { post4 }
end if;
```

--!post: a=a_i and b=b_i and m=(a+b)/2 and -100<=m and m<=100;

Automatic Program Provers (APP)

There exist some systems, other program verifiers work interactively.

(In comparison with the theoretical work there are quite few systems)

Freining/Kauer/Winkler 2002 compare 3 APP (by 26 examples)

FPP	:	Frege Program Prover (FSU Jena)
NPPV	:	New Paltz Program Verifier (SUNY / Marburg)
SPARK 6.0	:	SPADE Ada Real-Time Kernel, V 5.01, automatic + interactive

Feinerer 2005 compares 4 program provers (mostly usability)

FPP	:	Frege Program Prover (APP, FSU Jena)
KeY	:	interactive prover (Karlsruhe et al.)
Perfect Developer	:	APP (EscherTechnologies)
Prototype Verification System	:	interactive prover (SRI)

Rest

Mechanical Program Verification – Part 2

Jürgen F H Winkler
Institute of Informatics
Friedrich Schiller University
Jena, Germany

ELTE, Budapest, 25 – 29 Sep 2006

Overview

- Program Verification
- Historical Overview and Basic Concepts
- *Mechanical* Program Verification (MPV)
- Tools: FPP, NPPV, SPARK
- Problems with wp
- Summary
- References
- (Relational Approach
- Improved Adaptation Rule)

Mechanical Program Verification: Example

Better formula (improvement of Kahan's formula by Jürgen Winkler

mean(a, b) = if sig(a)=sig(b) …

everything OK ???

Unfortunately not: use of abs( ) $\Rightarrow$ domain must be symmetric interval

Yesterday evening it occurred to me that this is not completely true

Why ???

TYPE PosRangeTy IS range 20.0 .. 30.0;

In such a range abs(·) is the same as the identity function +(·)

Remark: +(·) is the only operation in the integer arithmetic of Java which always computes the mathematically correct result ;-)

Mechanical Program Verification: Example

Better formula (improvement of Kahan's formula by Jürgen Winkler

mean(a, b) = if sig(a)=sig(b) …

everything OK ???

Unfortunately not: use of abs( ) ⇒ domain must be symmetric interval

∨ domain ⊂ ℕ

Observe: this refers only to abs(·) and does not imply that the

Kahan-Winkler algorithm works in 20.0 .. 30.0