DE LA RECHERCHE À L'INDUSTRIE

Difference Logic with Floating Point numbers

SOPRANO | F.Bobot

05 Juin

www.cea.fr

INSTITUT CARNOT CEA LIST

digiteo

List

Range_Add

```
procedure
  Range_Add (X: Float_32; Res: out Float_32) is
   begin
      pragma Assume (X in 10.0 .. 1000.0);
      Res := X + 2.0;
      pragma Assert (Res >= 12.0);
   end Range_Add;
```

Range_Add

```
procedure
  Range_Add (X: Float_32; Res: out Float_32) is
   begin
      pragma Assume (X in 10.0 .. 1000.0);
      Res := X + 2.0;
      pragma Assert (Res >= 12.0);
   end Range_Add;
```

COLIBRI: UNSAT during propagation (Res in 12.0 .. 1002.0)

Range_Add

```
real_vars(float,[X,Res]),
X $: 10.0 .. 1000.0,
Res $= X + 2.0,
not(Res $>= 12.0) #= 1.
```

Range_Add

```
procedure
 Range_Mult (X : Float_32; Res : out Float_32) is
   begin
      pragma Assume (X in 5.0 .. 10.0);
      Res := X * 2.0 - 5.0;
      pragma Assert (Res >= X);
   end Range_Mult;
```

Range_Add

```
procedure
 Range_Mult (X : Float_32; Res : out Float_32) is
   begin
      pragma Assume (X in 5.0 .. 10.0);
      Res := X * 2.0 - 5.0;
      pragma Assert (Res >= X);
   end Range_Mult;
```

COLIBRI: UNSAT during propagation

- interval arithmetics OK for small domains
- propagation stops due to reduction threshold for big domains
- but deltas/difference logic for FP works here

Range_Add_Mult

```
procedure
   Range_Add_Mult (X, Y, Z : Float_32; Res : out Fl
   begin
      pragma Assume (X >= 0.0 and then X <= 180.0);
      pragma Assume (Y >= -180.0 and then Y <= 0.0)
      pragma Assume (Z >= 0.0 and then Z <= 1.0);
      pragma Assume (X + Y >= 0.0);
      Res := X + Y * Z;
      pragma Assert (Res >= 0.0 and then Res <= 360
   end Range_Add_Mult;
```

Range_Add_Mult

```
procedure
   Range_Add_Mult (X, Y, Z : Float_32; Res : out Fl
   begin
      pragma Assume (X >= 0.0 and then X <= 180.0);
      pragma Assume (Y >= -180.0 and then Y <= 0.0)
      pragma Assume (Z >= 0.0 and then Z <= 1.0);
      pragma Assume (X + Y >= 0.0);
      Res := X + Y * Z;
      pragma Assert (Res >= 0.0 and then Res <= 360
   end Range_Add_Mult;
```

COLIBRI: UNSAT during propagation (Res in 0.0 .. 180.0)

Guarded_Div

```
procedure
 Guarded_Div (X, Y : Float_32; Res : out Float_32)
   Threshold : constant Float_32 := 1000.0;
 begin
     pragma Assume (X >= 0.0);
     pragma Assume (Y > 0.1);
     pragma Assume (Y < 1.0);
     pragma Assume (X / Threshold <= Y);
     Res := X / Y;
     pragma Assert (Res < Threshold);
end Guarded_Div;
```

Guarded_Div

```
procedure
 Guarded_Div (X, Y : Float_32; Res : out Float_32)
   Threshold : constant Float_32 := 1000.0;
 begin
     pragma Assume (X >= 0.0);
     pragma Assume (Y > 0.1);
     pragma Assume (Y < 1.0);
     pragma Assume (X / Threshold <= Y);
     Res := X / Y;
     pragma Assert (Res < Threshold);
end Guarded_Div;
```

COLIBRI QUERY: SAT during labeling SAT : eg. X = 587.0, Y = 0.587, Res = 1000.0

Fibonacci

```
procedure
 Fibonacci (N : Positive; X, Y : Float_32;
 Res : out Float_32) is
begin
   pragma Assume (N in 2 .. 46);
   pragma Assume (X < (1.6181**(N-2))/2.2360 + 1.0);
   pragma Assume (Y < (1.6181**(N-1))/2.2360 + 1.0);
   Res := X + Y;
   pragma Assert (Res < (1.6181**N)/2.2360 + 1.0);
end Fibonacci;
```

Fibonacci

```
procedure
 Fibonacci (N : Positive; X, Y : Float_32;
 Res : out Float_32) is
begin
   pragma Assume (N in 2 .. 46);
   pragma Assume (X < (1.6181**(N-2))/2.2360 + 1.0);
   pragma Assume (Y < (1.6181**(N-1))/2.2360 + 1.0);
   Res := X + Y;
   pragma Assert (Res < (1.6181**N)/2.2360 + 1.0);
end Fibonacci;
```

COLIBRI QUERY (when asking to enumerate the values of N):

- SAT for each N in 2..21 (labeling)
- UNSAT for N in 22..46 (propagation)

Int_To_Float_Complex

```
procedure
 Int_To_Float_Complex (X : Unsigned_16; Y : Float_3
 Res : out Float_32) is
 S_Max   : constant := 10.0;  S_MSB   : constant :=
 S_Scale : constant := 2.0 ** 16 / S_MSB;
begin
   pragma Assume (Y in 0.25 .. 1.0);
   Res := Float_32 (X) / S_Scale;
   if Res >= S_Max then Res := Res - S_MSB; end if;
   Res := Res / Y;  --  overflow check unprovable
end Int_To_Float_Complex;
```

Int_To_Float_Complex

```
procedure
 Int_To_Float_Complex (X : Unsigned_16; Y : Float_3
 Res : out Float_32) is
 S_Max   : constant := 10.0;  S_MSB   : constant :=
 S_Scale : constant := 2.0 ** 16 / S_MSB;
begin
   pragma Assume (Y in 0.25 .. 1.0);
   Res := Float_32 (X) / S_Scale;
   if Res >= S_Max then Res := Res - S_MSB; end if;
   Res := Res / Y;  --  overflow check unprovable
end Int_To_Float_Complex;
```

COLIBRI QUERY: UNSAT

- during 3B filtering
- or 2B with refutations of #=> left hand side

2B/3B filtering

- 2B filtering: bound consistency.
- reduce extremum by 2B consistency

Int_To_Float_Simple

```
procedure
 Int_To_Float_Simple (X : Unsigned_16; Res : out Fl
    L : constant := 7.3526e6;
begin
    pragma Assume (X /= 0);
    pragma Assert (Float_32 (X) >= 0.9);  --  @ASSE
    Res := L / Float_32 (X);              --  @OVER
end Int_To_Float_Simple;
```

Int_To_Float_Simple

```
procedure
 Int_To_Float_Simple (X : Unsigned_16; Res : out Fl
    L : constant := 7.3526e6;
begin
    pragma Assume (X /= 0);
    pragma Assert (Float_32 (X) >= 0.9);  --  @ASSE
    Res := L / Float_32 (X);              --  @OVER
end Int_To_Float_Simple;
```

COLIBRI QUERY: UNSAT during propagation (Float_32(X) in 1.0 .. 65535.0)

Float_to_Long_Float

```
function Float_to_Long_Float (X : Float) return Lon
  Tmp : Long_Float;
begin
   pragma Assume (X >= Float'First
                  and X <= Float'Last);
   Tmp := Long_Float (X);
   pragma Assert
        (Tmp >= Long_Float (Float'First) and
         Tmp <= Long_Float (Float'Last));
   return Tmp;
end Float_to_Long_Float;
```

Float_to_Long_Float

```
function Float_to_Long_Float (X : Float) return Lon
  Tmp : Long_Float;
begin
   pragma Assume (X >= Float'First
                     and X <= Float'Last);
   Tmp := Long_Float (X);
   pragma Assert
         (Tmp >= Long_Float (Float'First) and
          Tmp <= Long_Float (Float'Last));
   return Tmp;
end Float_to_Long_Float;
```

COLIBRI QUERY : UNSAT during propagation

Difference Logic on Integers

A graph with:

- Vertexes are terms
- Edges are difference/distance between terms/vertex
- Presence of cycle with non null sum $\equiv$ Unsatisfiability

A float moved by another

$$d(a, c) = \text{“The number of float between } a \text{ and } c\text{”}$$

$$d(a, a + b) = a \xrightarrow{b}$$

Computation of $a \xrightarrow{b}$

Ex: $\mathcal{D}_a = [-\alpha; \alpha]$, $\mathcal{D}_b = \{\beta\}$ and $0 < \beta < \alpha$

$$a \xrightarrow{b} \in [m; M]?$$

Computation of $a \xrightarrow{b}$

Ex: $\mathcal{D}_a = [-\alpha; \alpha]$, $\mathcal{D}_b = \{\beta\}$ and $0 < \beta < \alpha$

$$a \xrightarrow{b} \in [m; M]?$$

$$M = d(-\frac{\beta}{2}, \frac{\beta}{2})$$

Computation of $a \xrightarrow{b}$

Ex: $\mathcal{D}_a = [-\alpha; \alpha]$, $\mathcal{D}_b = \{\beta\}$ and $0 < \beta < \alpha$

$$a \xrightarrow{b} \in [m; M]?$$

$$M = d(-\frac{\beta}{2}, \frac{\beta}{2})$$

$$m = d(\alpha, \alpha + \beta)$$

Example

$$5 \leq x \leq 10 \implies 2 * x - 5 \leq x$$

$$x + 1 = y \wedge y + 1 = x$$