Parameterized Gauge Functions for Numerical Precision Tuning in Dataflow Languages

Nasrine DAMOUCHE and Xavier Thirioux

IRIT/ENSEEIHT, ACADIE Team

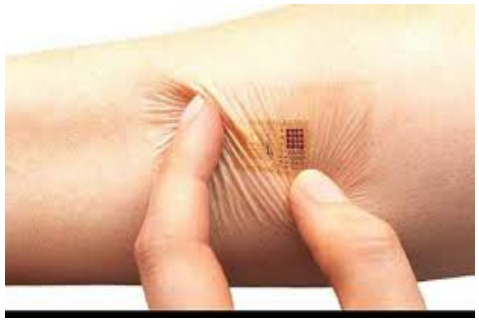
FEANICSES, Toulouse

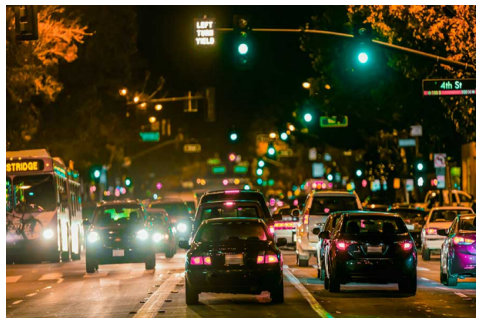
June 19-21, 2019

Context

Critical Embedded Systems

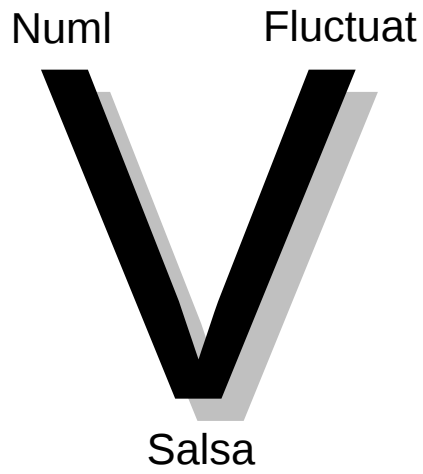







Context

- 1980': Birth of IEEE754 Standard
- 1990': First bugs due to IEEE754 Standard
  - Patriot rocket, Ariane 5, USS Yorktown ship
- 2000': First verification and validation tools
  - **Fluctuat** [ESOP'02], **Astrée** [ESOP'05]



- 2010': First Automatic Transformation Tools to Correct/Optimize
  - Accuracy: **Sardana** [SAS'12], **Herbie** [PLDI'15], **Salsa** [FMICS'15]
  - Precision tuning: **Precimonious** [SC'13], **Rosa** [POPL'14]
- 2020': Assisted developement tools: **Numl** [ICFEM'18], **Pretty !?**

Mixed Precision (NUML [ICFEM'18])

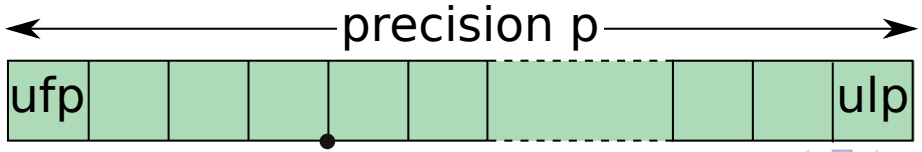
- **NUML** : System type for floating-point computations using
  - Dependent types and type inference algorithm
- **Input** : An expression + (sign, ufp, precision)
- **Output** : Types encoding accuracy

$$\text{type } \texttt{real}\{s, u, p\}$$

- s sign of x, $s \in \{+, -, 0, *\}$
- u $\mathsf{ufp}(x)$, i.e. $u$nit in the $f$irst $p$lace

$$\mathsf{ufp}(x) = min\{i \in \mathbb{N} : 2^{i+1} > x\}$$

- p precision (number of bits/digits of x)



$$err \leq 2^{u-p+1}$$

Mixed Precision (NUML [ICFEM'18])

- **Typing rules**

$$\frac{}{\Gamma \vdash \texttt{i} : \texttt{int}} \ (\textsc{Int}) \qquad \frac{}{\Gamma \vdash \texttt{b} : \texttt{bool}} \ (\textsc{Bool})$$

$$\frac{\mathsf{sign}(\texttt{r}) \prec s \quad \mathsf{ufp}(\texttt{r}) \leq u}{\Gamma \vdash \texttt{r\{s,u,p\}} : \texttt{real\{s,u,p\}}} \ (\textsc{Real}) \qquad \frac{\Gamma(\texttt{id}) = t}{\Gamma \vdash \texttt{id} : t} \ (\textsc{Id})$$

$$\frac{\Gamma \vdash e_0 : \texttt{bool} \qquad \Gamma \vdash e_1 : t_1 \qquad \Gamma \vdash e_2 : t_2 \qquad t = t_1 \sqcup t_2}{\Gamma \vdash \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 : t} \ (\textsc{Cond})$$

$$\frac{\Gamma, x : t_1 \vdash e : t_2}{\Gamma \vdash \lambda x.e : \Pi x : t_1.t_2} \ (\textsc{Abs}) \qquad \frac{\Gamma, x : t_1, f : \Pi.y : t_1.t_2 \vdash e : t_2 \quad y \text{ not free in } t_2}{\Gamma \vdash \texttt{rec } f \ x.e : \Pi x : t_1.t_2} \ (\textsc{Rec})$$

$$\frac{\Gamma \vdash e_1 : \Pi x : t_0.t_1 \qquad \Gamma \vdash e_2 : t_2 \qquad t_2 \sqsubseteq t_0}{\Gamma \vdash e_1 \ e_2 : t_1[x \mapsto e_2]} \ (\textsc{App})$$

Some Examples with NUML

Accuracy of the Results Explicited

```
> let f = fun x -> x + 1.0 ;;
val f :  'a,'b,'c -> <expr>,<expr>,<expr> = <fun>

> f 1.234 ;;
- :  +,1,53 = 2.2340000000000000 +/- 2.22044604925e-16

> f 1.234{4} ;;
- :  +,1,4 = 2.23 +/- 0.0625


> (1.0e15 + 1.0) - 1.0e15 ;;
- :  +,50,54 = 1.0 +/- 0.0625

> (1.0e16 + 1.0) - 1.0e16 ;;
Error:  The computed value has no significant digit.  Its ufp is 0 but the ulp of the
certified value is 1
```

Some Examples with NUML

Recursivity and Subtyping

```
> let rec g x = if x < 1.0 then x else g (x * 0.07) ;;
val g :  +,0,53 -> +,0,53 = <fun>

> g 1.0;;
- :  +,0,53 = 0.070000000000000 +/- 1.11022302463e-16

> g 2.0 ;;
Error:  This expression has type +,1,53 but an expression was expected of type +,0,53
```

$$s_1, u_1, p_1 \sqsubseteq s_2, u_2, p_2 \iff s_1 \preceq s_2, \quad u_1 \leq u_2, \quad p_1 \geq p_2 + u_1 - u_2$$

Some Examples with NUML

Other Examples

$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \ldots$

```
> let rec taylor x{-1,25} xn i n =
              if (i>n) then 0.0{*,10,20}
              else xn + (taylor x (x*xn) (i+_ 1) n) ;;
val taylor :  *,-1,25 -> *,10,20 -> int -> int -> *,10,20 = <fun>

> taylor 0.2 1.0 0 5;;
- :  *,10,20 = 1.2499 +/- 0.0009765625
```

```
> let discriminant a b c = b × b - 4.0 × a × c ;;
 val discriminant : real{'a,'b,'c} -> real{'d,'e,'f} -> real{'g,'h,'i} ->
real{<expr>,<expr>,<expr>} = <fun>

> discriminant 2.0 -11.0 15.0 ;;
- : real{+,8,52} = 1.000000000000
```

Current Issues

- Many useful programs cannot be typed
- Due to potential "divergence" (not statically bounded values)
- Problem: very frequent in embedded systems (e.g. PID controllers)

Example: y integrates input x

```
while (...) {
 x=input();
 y=y+x;
}
```

- Even if x is bounded, y may diverge and so its ufp

Current Issues

- Many useful programs cannot be typed
- Due to potential "divergence" (not statically bounded values)
- Problem: very frequent in embedded systems (e.g. PID controllers)

Example: y integrates input x

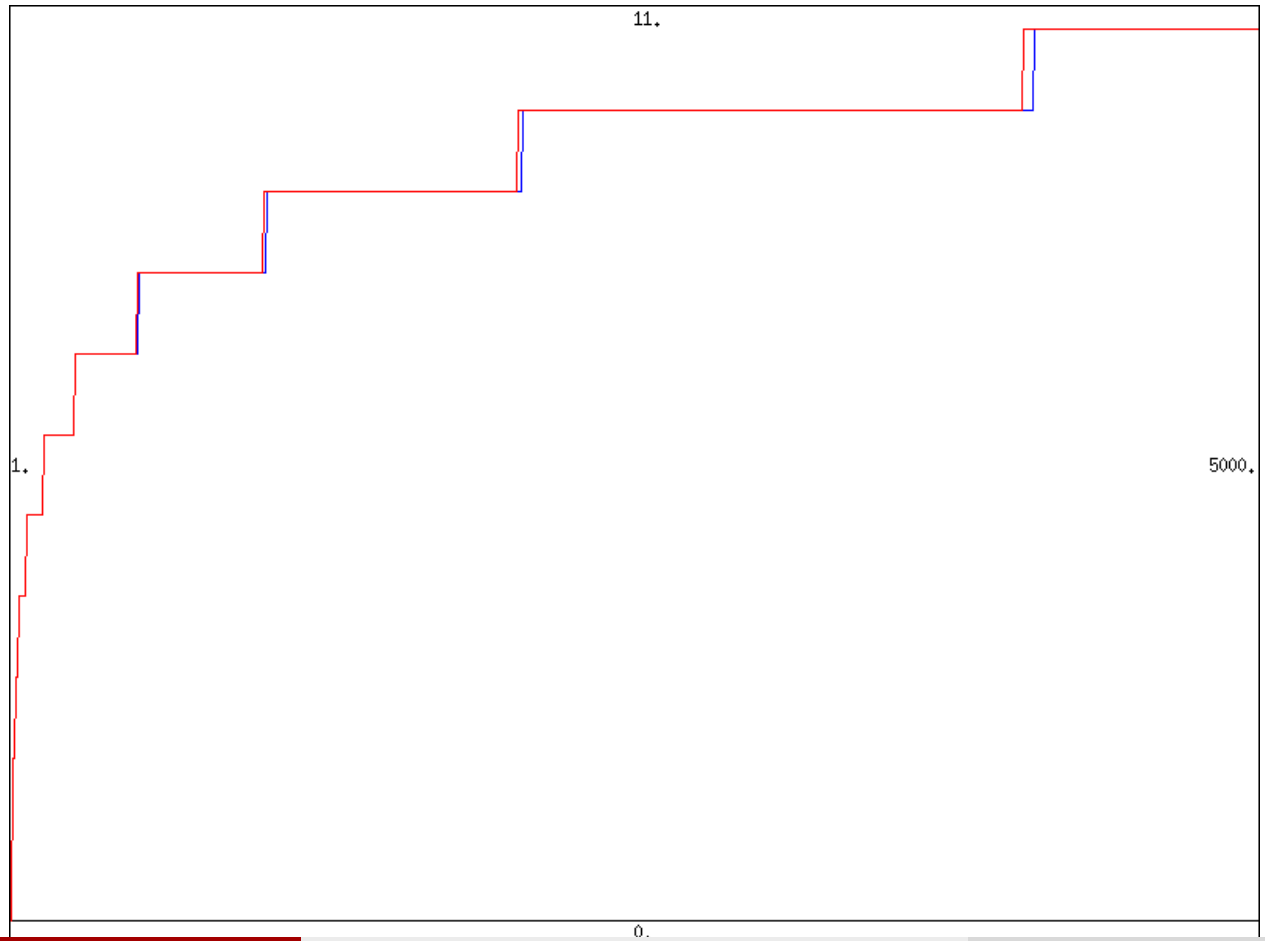
```
while (...) {
 x=input();
 y=y+x;
}
```

- Even if x is bounded, y may diverge and so its ufp
- But how fast ? Can we specify divergence speed ?

Example 1

Example (5000 iterations)

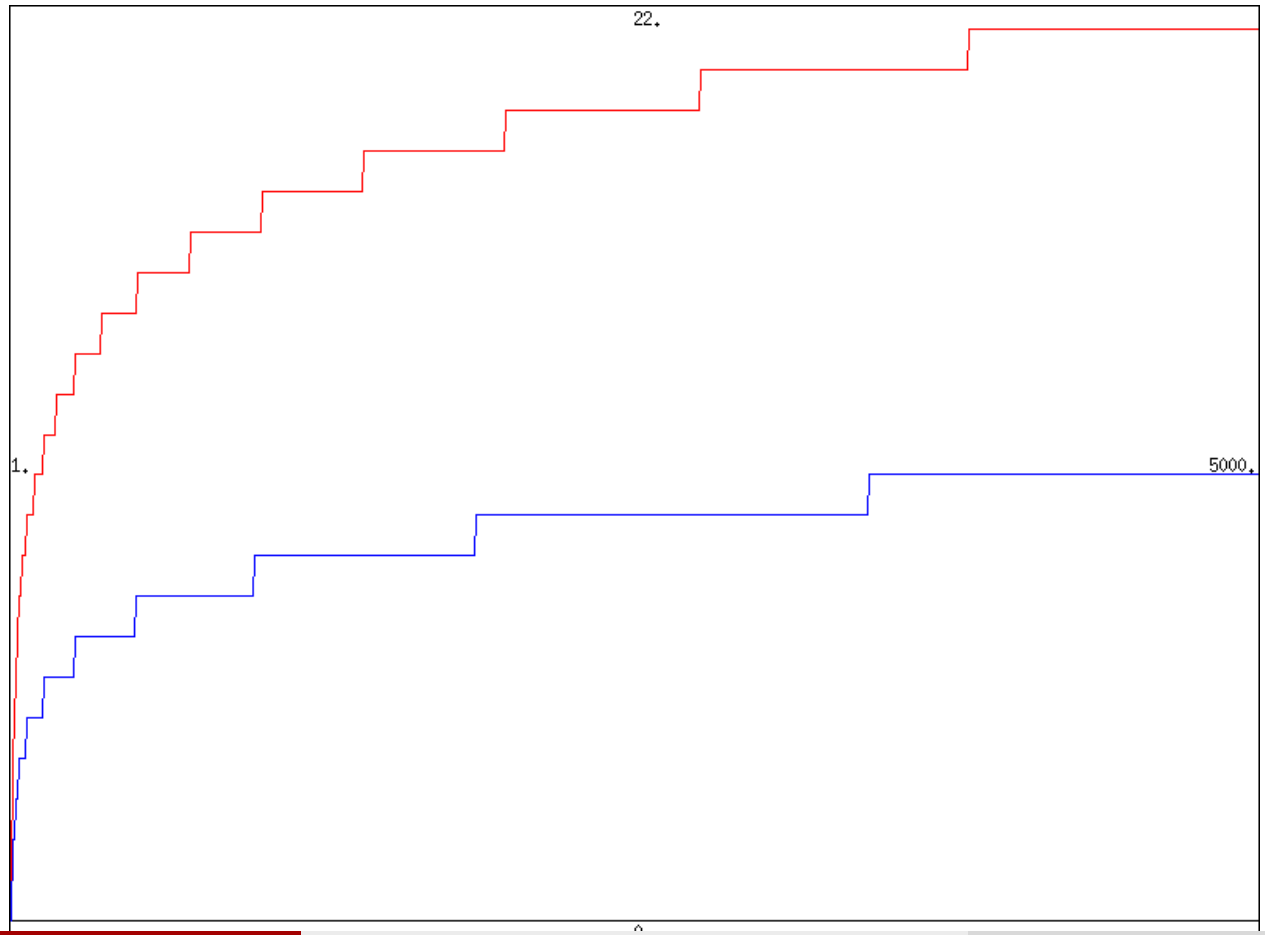
- x = x + 0.5 ;
- y = 0.01 * y + x ;



Example 2

Example (5000 iterations)

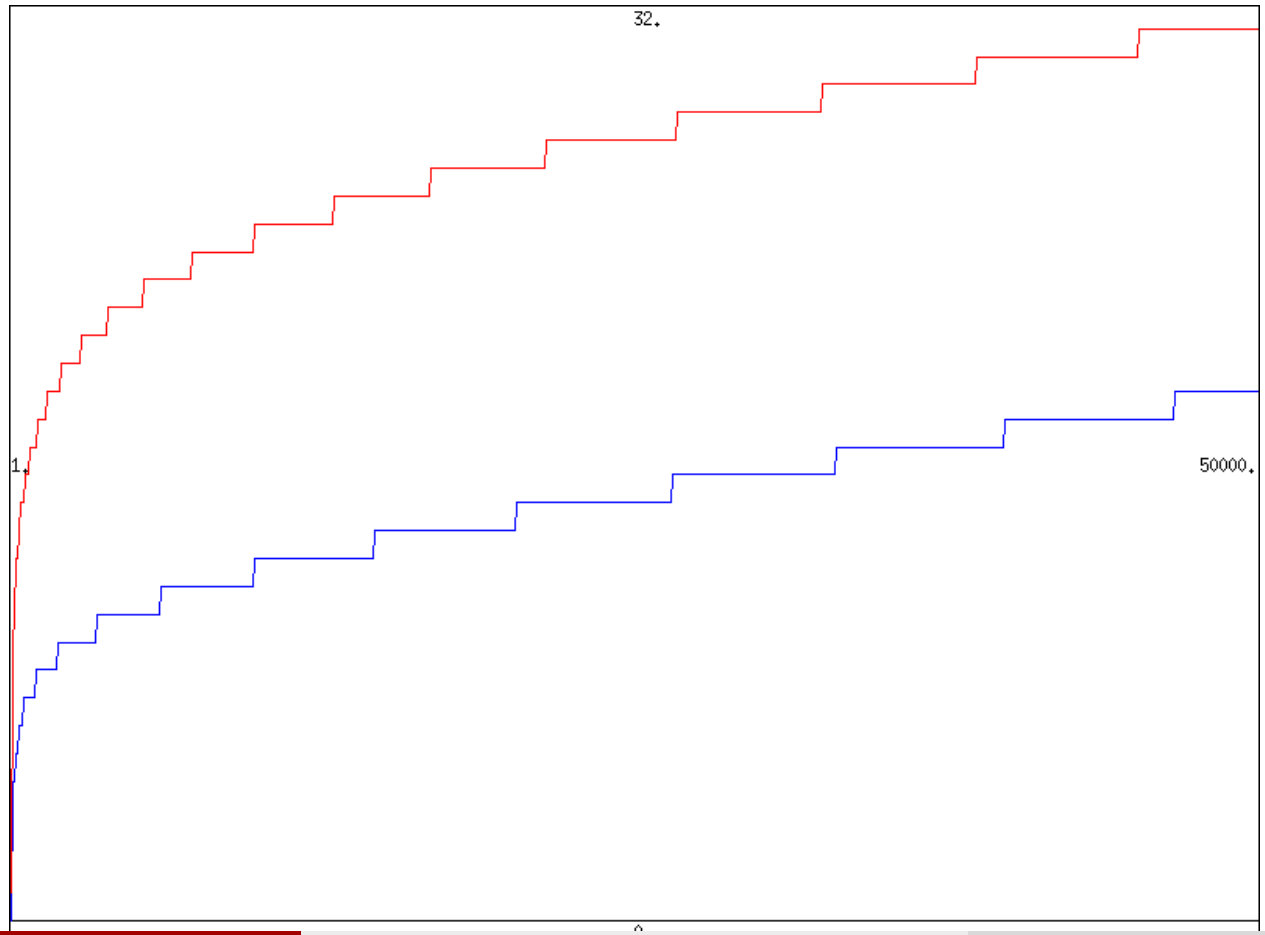
- x = 1.0001 * x + 0.5 ;
- y = y + x ;



Example 2

Example (50000 iterations)

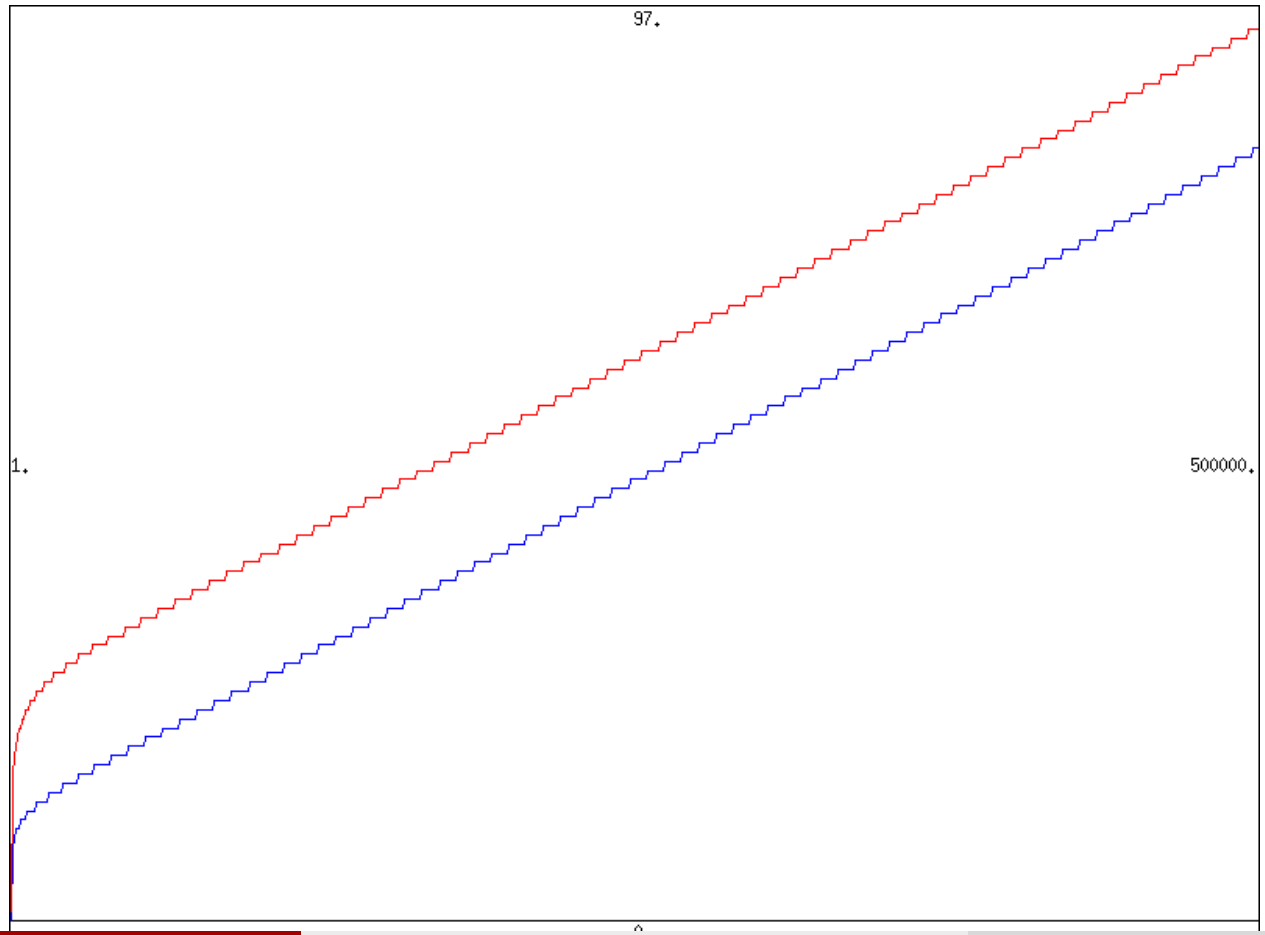
- x = 1.0001 * x + 0.5 ;
- y = y + x ;



Example 2

Example (500000 iterations)

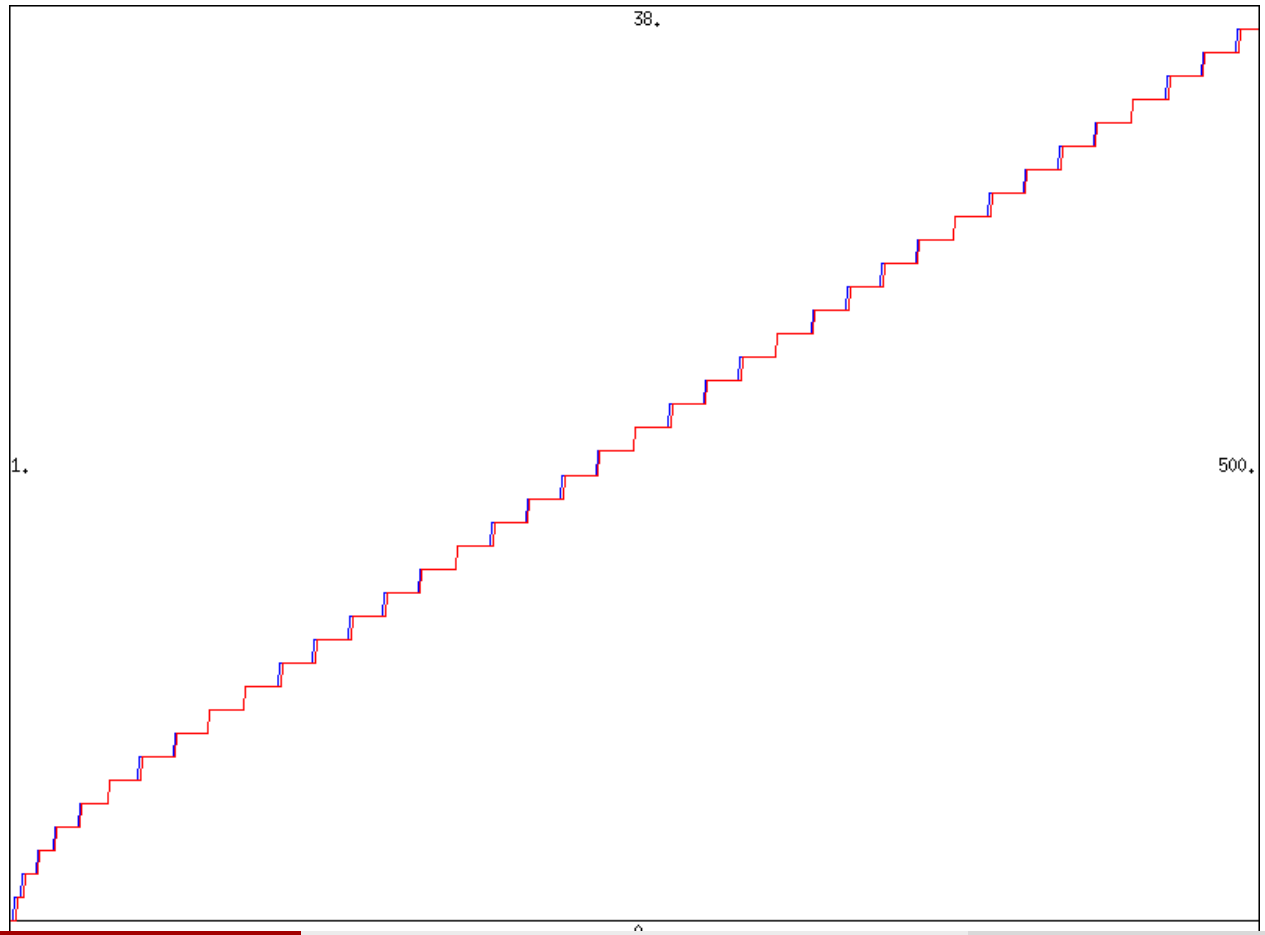
- x = 1.0001 * x + 0.5 ;
- y = y + x ;



Example 3

Example (500 iterations)

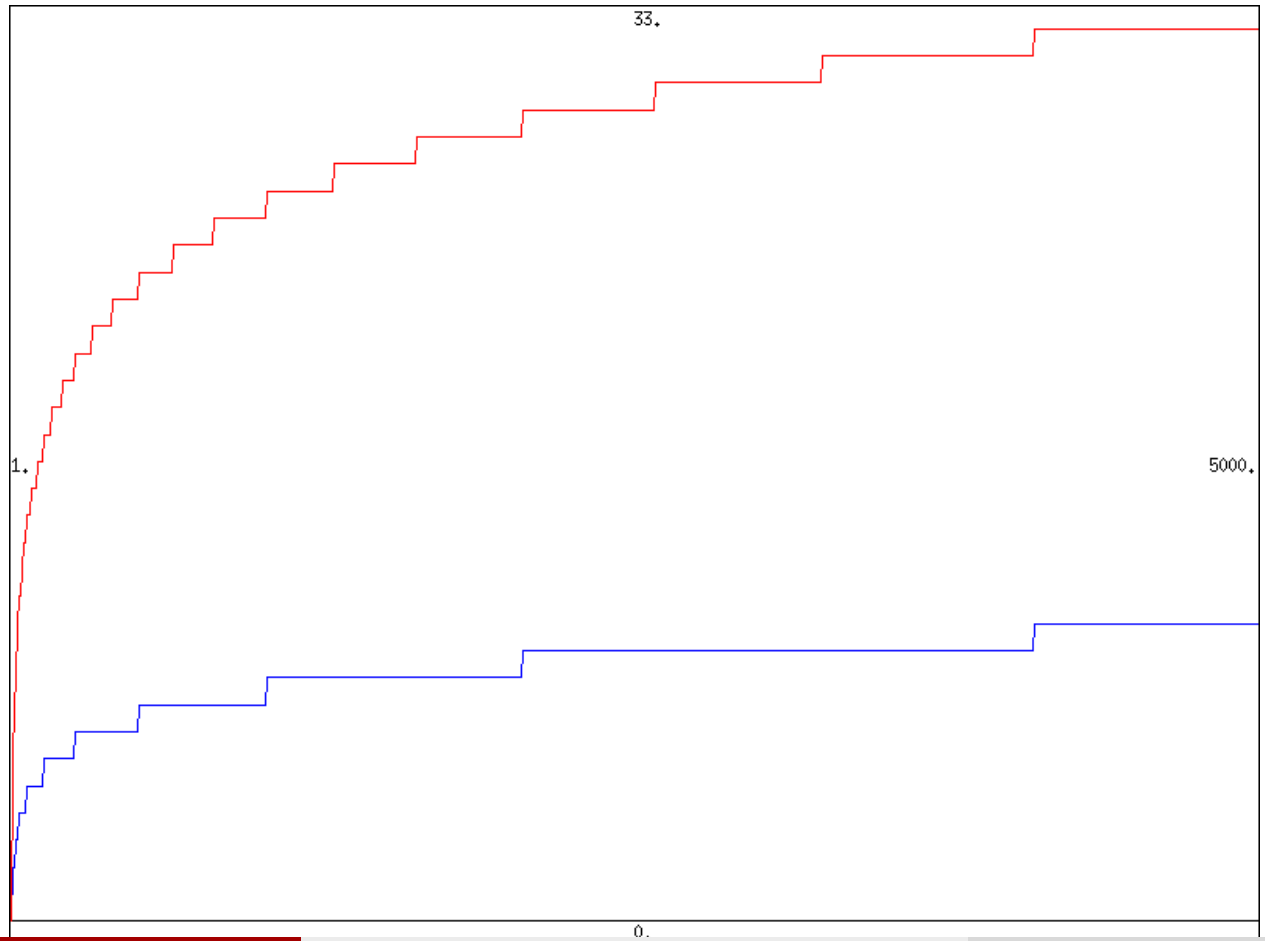
- x = 1.05 * x + 0.5 ;
- y = 0.01 * y + x ;



Example 4

Example (5000 iterations)

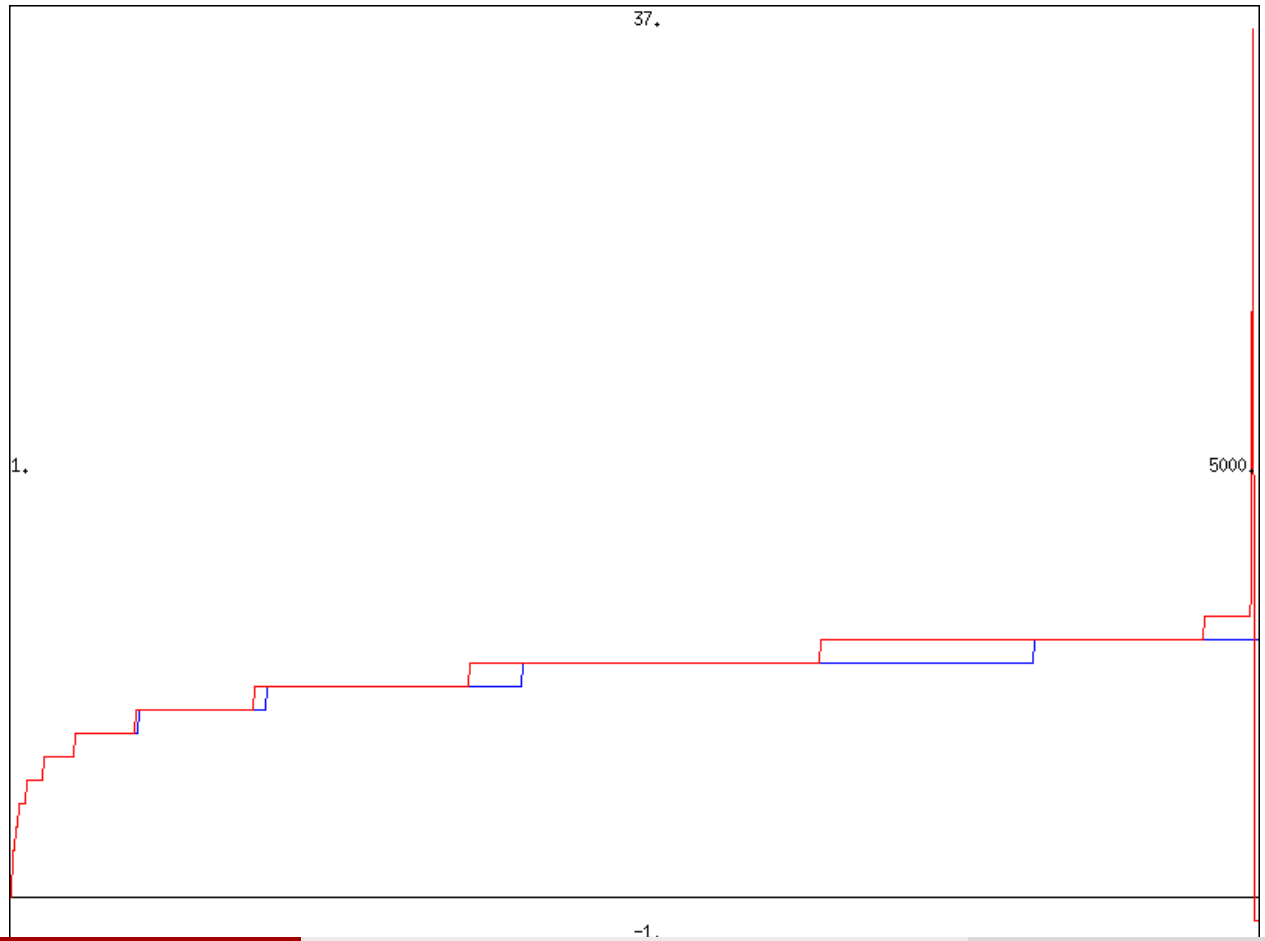
- x = x + 0.5 ;
- y = x*x*x ;



Example 5

Example (5000 iterations)

- x = x + 0.5 ;
- y = 0.0001019 * y * y + x ;



Synthesis

- Pervasive log-diverging ufp w.r.t. iteration number (so it seems)
- Integration preserves log-divergence
- log-diverging ufp doesn't grow that fast
- Sweet spot: (simple) control loop bodies
- Target language for compiling synchronous languages

Main Contribution

- Determining the **ufp** of arithmetic expressions/instructions
- Searching an **upper bound of the ufp** in **quasi-log-affine form**
- Expressing solutions in symbolic form:

$$\text{ufp}(t') \leq k_0 + k_1 log_2(1+t) + \frac{k_2}{1+t} \mid 0 \leq t' \leq t \in \mathbb{N}; \quad k_0, k_1, k_2 \in \mathbb{R}$$

Type System

- Grammar of input language (control loop bodies)

$$
\begin{aligned}
\text{instr} \quad ::= & \; l \leftarrow K \\
& \mid l \leftarrow x + y \\
& \mid l \leftarrow x - y \\
& \mid l \leftarrow x * y \\
& \mid s \leftarrow l \\
& \mid instr; instr
\end{aligned}
$$

assuming

- l : local variable
- s : state variable
- instructions normalized and in SSA form

Type System

- A typing rule for each instruction
- A triple $\langle k_0, k_1, k_2 \rangle$ for the ufp of each variable
- ufp of variables are such that $k_1 \geq 0$
- Only ufp of local variables may exhibit a $\frac{k_2}{1+t}$ component with $k_2 \neq 0$
- Solutions in symbolic form, as a constraint system
- General form of linear constraint (shortened as $\langle k_0, k_1, k_2 \rangle \geq 0$) :

$$\frac{k_0 + k_1 \geq 0, \quad k_0 + k_2 \geq 0, \quad k_1 \geq 0}{k_0 + k_1 \log_2(1+t) + \frac{k_2}{1+t} \geq 0}$$

Typing rules

instruction	constraint set
$l \leftarrow K$:	$\{\langle l_0 - \mathrm{ufp}(K), l_1, l_2 \rangle \geq 0\}$
$l \leftarrow x \pm y$:	$\{\langle x_0 - y_0, x_1 - y_1, x_2 - y_2 \rangle \geq 0,$ $\langle l_0 - x_0, l_1 - x_1, l_2 - x_2 - 1 \rangle \geq 0,$ $\langle 0, x_1 - y_1 - 1, 0 \rangle \geq 0\}$
$l \leftarrow x * y$:	$\{\langle l_0 - x_0 - y_0 - 1, l_1 - x_1 - y_1, l_2 - x_2 - y_2 \rangle \geq 0\}$
$s \leftarrow l$:	$\{\langle s_0 - l_0, s_1 - l_1, \frac{s_1}{2\log(2)} - l_2 \rangle \geq 0, s_2 = 0\}$

Typing rule for addition/subtraction

Where we need to handle separate solution sets (no principal typing)

- Basic rule for $l \leftarrow x \pm y$: $\text{ufp}_l(t) \geq max(\text{ufp}_x(t), \text{ufp}_y(t)) + 1$
- Better amortized rule: replace $+1$ (carry out) by $+2^{-|\text{ufp}_x(t) - \text{ufp}_y(t)|}$
- 2 separate cases w.r.t. $\text{ufp}_x(t) \geq \text{ufp}_y(t)$
- The first case yields: $\text{ufp}_l(t) \geq \text{ufp}_x(t) + 2^{-\text{ufp}_y(t) - \text{ufp}_x(t)}$

$\Leftrightarrow\ l_0 + l_1 * \log_2(1+t) + \frac{l_2}{1+t} \geq x_0 + x_1 \log_2(1+t) + \frac{x_2}{1+t} + 2^{y_0 - x_0}(1+t)^{y_1 - x_1} 2^{\frac{y_2 - x_2}{1+t}}$

$\Leftarrow\ (l_0 - x_0) + (l_1 - x_1) * \log_2(1+t) + \frac{l_2 - x_2}{1+t} \geq 2^{y_0 - x_0 + y_2 - x_2} * (1+t)^{y_1 - x_1}$

$\Leftarrow\ (l_0 - x_0) + (l_1 - x_1) * \log_2(1+t) + \frac{l_2 - x_2 - 1}{1+t} \geq 0,\ x_1 - y_1 \geq 1$

- Finally, as a constraint set:
  $\{\langle x_0 - y_0, x_1 - y_1, x_2 - y_2 \rangle \geq 0, \langle l_0 - x_0, l_1 - x_1, l_2 - x_2 - 1 \rangle \geq 0, x_1 - y_1 - 1 \geq 0\}$

Typing rule for state variable assignment

Where we need $\frac{k_2}{1+t}$ component

- State assignment $s \leftarrow l$ means: $\text{ufp}_s(t+1) \geq \text{ufp}_l(t)$
- That is: $s_0 + s_1 \log_2(2+t) \geq l_0 + l_1 \log_2(1+t) + \frac{l_2}{1+t}$, assuming $s_2 = 0$

$\Leftarrow (s_0 - l_0) + (s_1 - l_1) \log_2(1+t) + s_1 \log_2(2+t) - s_1 \log_2(1+t) - \frac{l_2}{1+t} \geq 0$

$\Leftarrow (s_0 - l_0) + (s_1 - l_1) \log_2(1+t) + (\frac{s_1}{2\log(2)} - l_2)\frac{1}{1+t} \geq 0$

- Finally, as a constraint set: $\{\langle s_0 - l_0, s_1 - l_1, \frac{s_1}{2\log(2)} - l_2 \rangle \geq 0, s_2 = 0\}$

Typing an example

1. Original loop body:

$$x = x + 0.5;$$
$$y = y + x;$$

2. Target program:

$$l \leftarrow 0.5;$$
$$m \leftarrow x + l;$$
$$x \leftarrow m;$$
$$n \leftarrow y + x;$$
$$y \leftarrow n;$$

Typing an example

1. Constraint set (positivity of quasi-log-affine forms):

$$\begin{aligned}
\langle l_0 - \mathrm{ufp}(0.5), l_1, l_2 \rangle &\geq 0 \\
\langle x_0 - l_0, x_1 - l_1, x_2 - l_2 \rangle &\geq 0 \\
\langle m_0 - x_0, m_1 - x_1, m_2 - x_2 - 1 \rangle &\geq 0 \\
\langle 0, x_1 - l_1 - 1, 0 \rangle &\geq 0 \\
\langle x_0 - m_0, x_1 - m_1, \frac{x_1}{2\log(2)} - m_2 \rangle \geq 0, x_2 &= 0 \\
\langle y_0 - x_0, y_1 - x_1, y_2 - x_2 \rangle &\geq 0 \\
\langle n_0 - y_0, n_1 - y_1, n_2 - y_2 - 1 \rangle &\geq 0 \\
\langle 0, y_1 - x_1 - 1, 0 \rangle &\geq 0 \\
\langle y_0 - n_0, y_1 - n_1, \frac{y_1}{2\log(2)} - n_2 \rangle \geq 0, y_2 &= 0
\end{aligned}$$

Typing an example

1. Normalized constraint set on coefficients:

$$l_0 + 1 + l_1 \geq 0, l_0 + 1 + l_2 \geq 0, l_1 \geq 0$$
$$x_0 - l_0 + x_1 - l_1 \geq 0, x_0 - l_0 + x_2 - l_2 \geq 0, x_1 - l_1 \geq 0$$
$$m_0 - x_0 + m_1 - x_1 \geq 0, m_0 - x_0 + m_2 - x_2 - 1 \geq 0, m_1 - x_1 \geq 0$$
$$x_1 - l_1 - 1 \geq 0$$
$$x_0 - m_0 + x_1 - m_1 \geq 0, x_0 - m_0 + \frac{x_1}{2\log(2)} - m_2 \geq 0, x_2 = 0, x_1 - m_1 \geq 0$$
$$y_0 - x_0 + y_1 - x_1 \geq 0, y_0 - x_0 + y_2 - x_2 \geq 0, y_1 - x_1 \geq 0$$
$$n_0 - y_0 + n_1 - y_1 \geq 0, n_0 - y_0 + n_2 - y_2 - 1 \geq 0, n_1 - y_1 \geq 0$$
$$y_1 - x_1 - 1 \geq 0$$
$$y_0 - n_0 + y_1 - n_1 \geq 0, y_0 - n_0 + \frac{y_1}{2\log(2)} - n_2 \geq 0, y_2 = 0, y_1 - n_1 \geq 0$$

Typing an example

1. Simplified constraint set on coefficients:

$$
\begin{aligned}
x_2 = y_2 = 0, x_0 = m_0, x_1 = m_1, y_0 = n_0, y_1 = n_1 \\
l_0 + 1 + l_1 \geq 0, l_0 + 1 + l_2 \geq 0, l_1 \geq 0 \\
x_0 - l_0 + x_1 - l_1 \geq 0, x_0 - l_0 - l_2 \geq 0 \\
m_2 - 1 \geq 0 \\
x_1 - l_1 - 1 \geq 0 \\
\frac{x_1}{2\log(2)} - m_2 \geq 0 \\
y_0 - x_0 \geq 0 \\
n_2 - 1 \geq 0 \\
y_1 - x_1 - 1 \geq 0 \\
\frac{y_1}{2\log(2)} - n_2 \geq 0
\end{aligned}
$$

Typing an example

1. Minimizing slopes x_1, y_1, l_1 (and m_2, n_2):

$$x_2 = y_2 = 0, x_0 = m_0, x_1 = m_1, y_0 = n_0, y_1 = n_1$$
$$l_1 = 0, x_1 = 1, y_1 = 2, m_2 = n_2 = 1$$
$$l_0 + 1 \geq 0, l_0 + 1 + l_2 \geq 0$$
$$x_0 - l_0 + 1 \geq 0, x_0 - l_0 - l_2 \geq 0$$
$$y_0 - x_0 \geq 0$$

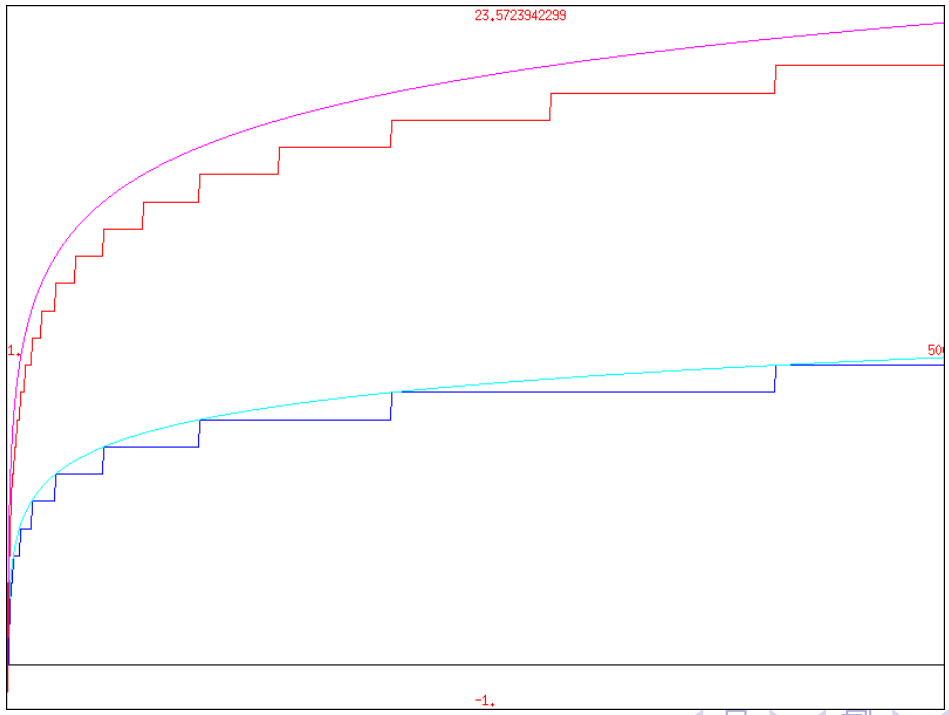
2. Then also minimizing x_0, y_0, l_0 (and l_2):

$$x_2 = y_2 = 0, x_0 = m_0, x_1 = m_1, y_0 = n_0, y_1 = n_1$$
$$l_1 = l_2 = 0, x_1 = 1, y_1 = 2, m_2 = n_2 = 1$$
$$y_0 = x_0 = l_0 = -1$$

3. If we had initial values, x_0, y_0 would have been further constrained

Typing an example

- Computed types versus reality:
  - computed $\mathrm{ufp}_x(t) = -1 + \log_2(1+t)$
  - computed $\mathrm{ufp}_y(t) = -1 + 2\log_2(1+t)$
  - real $\mathrm{ufp}_x(t)$
  - real $\mathrm{ufp}_y(t)$
- Graphically:



Conclusion

Current Work

- Implementing type system for synchronous programs

Future Work

- Experimenting with the prototype on real examples
- Proving correctness
- Experimenting with wider classes of ufp functions