Floating-Point Mixed Precision Tuning by Static Analysis

FEANICSES 2019

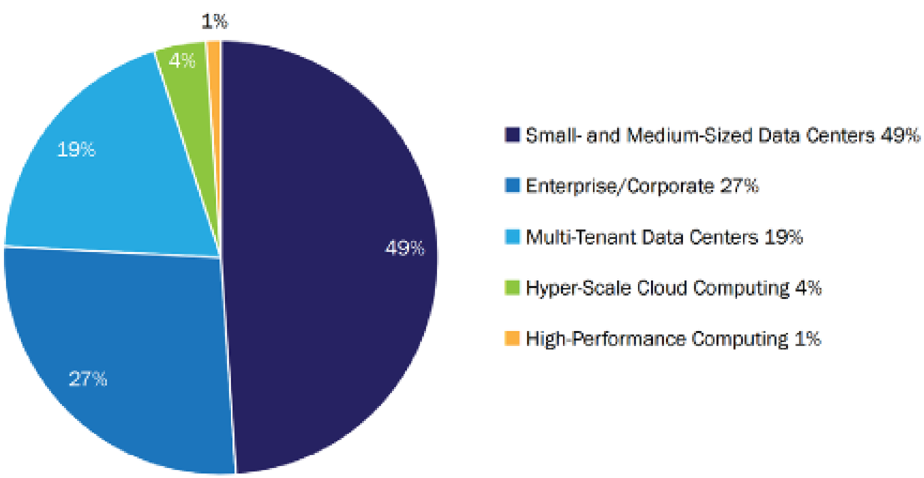
Dorra Ben Khalifa
Supervisors : Matthieu Martel & Assalé Adjé

University of Perpignan

21 juin 2019

Energy Consumption Concern

- *Top 500 supercomputers energy consumption* $\simeq$ ***$400 million/year***
- How to increase **energy efficiency ?**
  - **Green Computing :** *http ://www.green500.org*
  - Reduce application energy consumption
  - Sacrifice **accuracy** for **performance** $\Rightarrow$ **Floating-point precision tuning**



Estimated U.S. data center electricity consumption by market segment http://www.nrdc.org/

What About...

- Computer architectures support multiple levels of precision
  - **Higher precision :** improves accuracy
  - **Lower precision :** reduces energy, running time and bandwidth capacity
- Automatically tune floating-point precision is challenging
  - Without affecting correctness
  - Improving performance

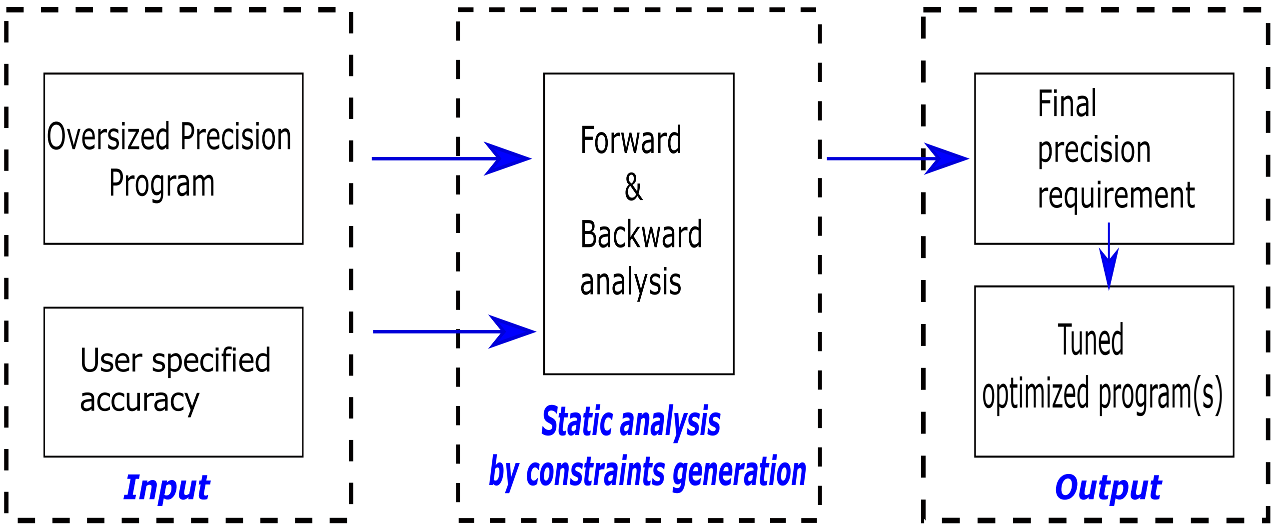
Precision vs Accuracy !

- ***Precision*** : number of bits representing a value (its format)
- ***Accuracy*** : how close a floating-point computation comes to the real value !

Related Work

- **TWIST**
  Static analysis by constraints generation
  TWIST [3]
- **CRAFT, Precimonious/HiFPTuner**
  Search based methods
  CRAFT [Lam'13 et al.], Precimonious/HiFPTuner [Rubio'13 et al.] [2]
- **FPTuner, Rosa/Daisy**
  Rigorous error analysis methods
  FPTuner [Chiang'17 et al.], Rosa/Daisy [Darulova'14 et al.]
- **Herbie, Salsa**
  Automatically discovering unstable floating-point operations and applying transformations
  Herbie [Panchekha'14 et al.], Salsa [DM18]

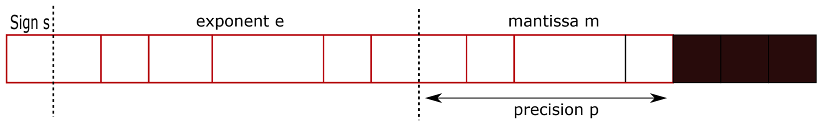
Overview of our Approach



Outline

Outline

Basic Concepts on Floating-Point Numbers



- A **floating-point number** x in base β :

$$x = s.m.\beta^{e-p+1}$$

  - **s** the sign, **m** the mantissa, **e** the exponent encoded in the bit string and **p** is the format precision
- **IEEE-754 Formats**

format	bit width	mantissa size (p - 1)	exponent size	bias
binary16	16	10	5	15
binary32	32	23	8	27
binary64	64	52	11	1023
binary128	128	112	15	16383

Ufp and *Ulp* Functions

Weight of the most significant bit :

$$ufp(x) = \min\{i \in \mathbb{N} : 2^{i+1} > x\} = \lfloor \log_2(x) \rfloor$$

Weight of the least significant bit :

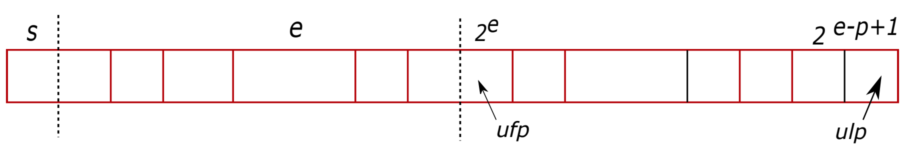
$$ulp(x) = \begin{cases} e - p & \text{round to nearest,} \\ e + 1 - p & \text{otherwise.} \end{cases}$$

$\mathbb{F}_p$: **Set of floating point numbers :** $|v - \widehat{v}| \leq 2^{e-p+1}$

$$\forall x \in \mathbb{F}_p, \quad ulp(x) = ufp(x) - p + 1$$

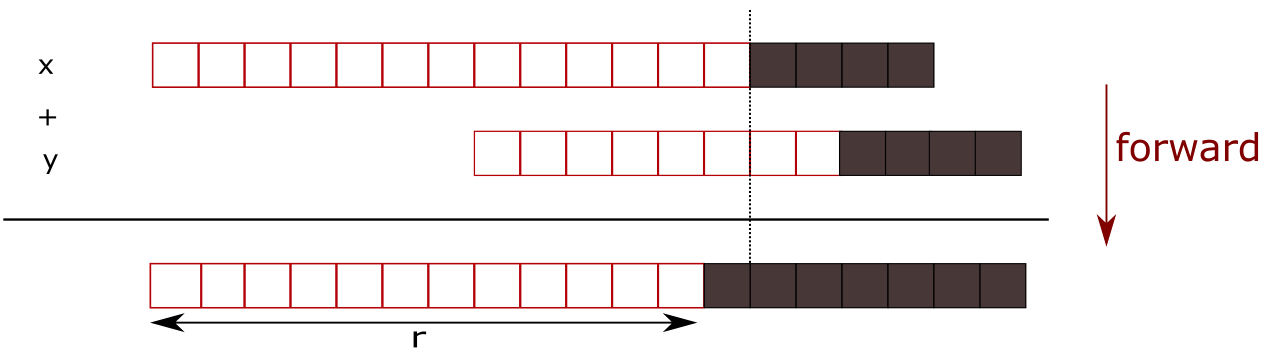
Error on x :

$$\epsilon(x) \leq 2^{ulp(x)}$$

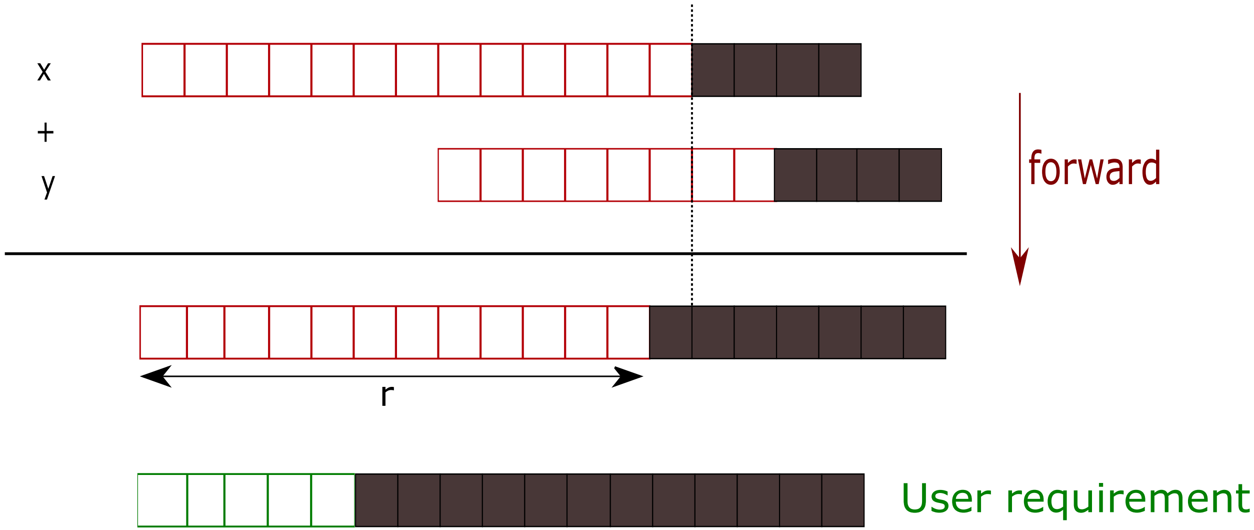


Outline

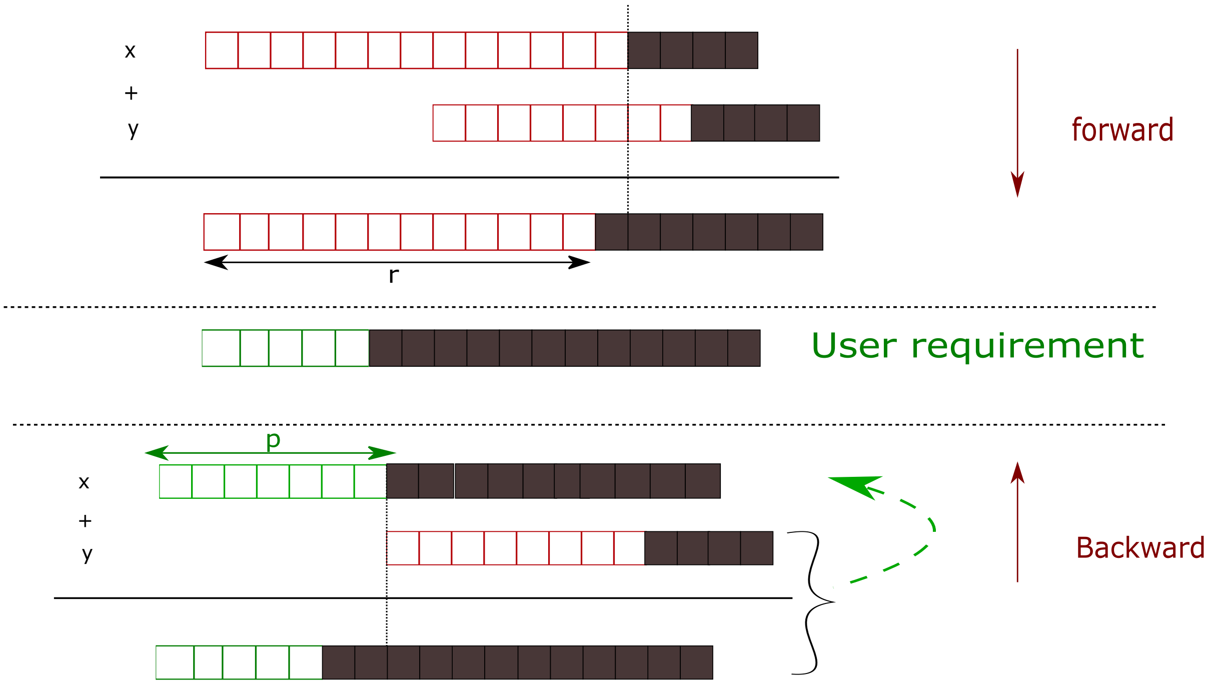
Forward & backward Analysis for Arithmetic Expressions



Forward & backward Analysis for Arithmetic Expressions



Forward & backward Analysis for Arithmetic Expressions



Generalizable technique into sets of values !

Abstract Domain

- **Abstract Values :** $[a, b]_p$ interval of $\mathbb{F}_p$

  e.g : $x, y \in [1.0, 3.0]_{16}, |v - \widehat{v}| \leq 2^{ufp(x)-15}$

- **Concretization function :**

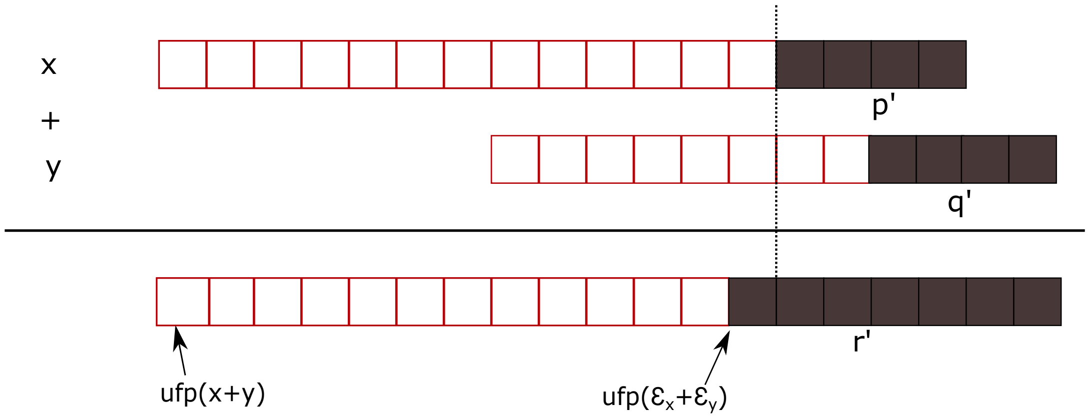
$$\gamma([a, b]_p) = x \in \mathbb{F}_p : a \leq x \leq b$$

- **Partial order :**

$$[a, b]_p \sqsubseteq [c, d]_q \Leftrightarrow [a, b] \subseteq [c, d] \wedge q \leq p$$

$[a, b]_p$ is more precise than $[c, d]_q$ with a greater accuracy

Concrete Addition in $\mathbb{F}_p$



- **Forward addition :**p' : size of ϵ_x, q' : size of ϵ_y

$$\overrightarrow{\oplus}(x_{p_{p'}}, y_{q_{q'}}) = z_{r_{r'}} \quad with \quad r = ufp(x+y) - ufp(\epsilon(x) + \epsilon(y))$$

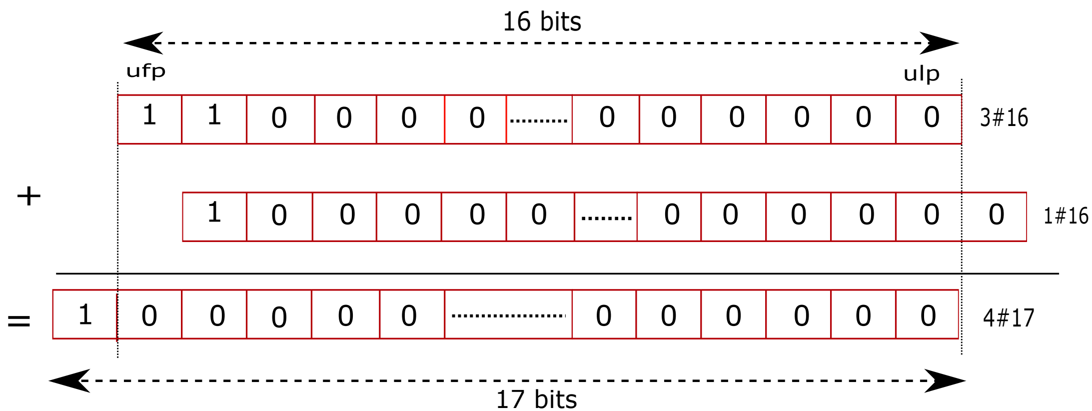
- **Backward addition :**

$$\overleftarrow{\oplus}(z_{r_{r'}}, y_{q_{q'}}) = (z-y)_{p_{p'}} \quad avec \quad p = ufp(z-y) - ufp(\epsilon(z) - \epsilon(y))$$

Abstract Addition in $\mathbb{I}_p$

$$\overrightarrow{\boxplus}([1.0, 3.0]_{16}, [1.0, 3.0]_{16}) = [2.0, 6.0]_{16}$$

$$\overleftarrow{\boxplus}([2.0, 6.0]_{10}, [1.0, 3.0]_{16}) = [1.0, 3.0]_{9}$$



$$\overrightarrow{\oplus}(1.0_{16}, 1.0_{16}) = 2.0_{16} \quad \overrightarrow{\oplus}(1.0_{16}, 3.0_{16}) = 4.0_{17}$$

$$\overrightarrow{\oplus}(3.0_{16}, 1.0_{16}) = 4.0_{17} \quad \overrightarrow{\oplus}(3.0_{16}, 3.0_{16}) = 6.0_{16}$$

Concrete Multiplication in $\mathbb{F}_p$

- **Forward multiplication :**

$$\overrightarrow{\otimes}(x_{p_{p'}}, y_{q_{q'}}) = z_{r_{r'}} \quad \textit{where} \quad r = ufp(x \times y) - ufp(\epsilon(x \times y))$$

$$\textit{and} \quad ufp(\epsilon(x \times y)) = y.\epsilon(x) + x.\epsilon(y) + \epsilon(x).\epsilon(y)$$

- **Backward multiplication :**

$$\overleftarrow{\otimes}(z_{r_{r'}}, y_{q_{q'}}) = (z \div y)_{r_{r'}} \quad \textit{where} \quad p = ufp(z \div y) - ufp\left(\frac{y.\epsilon(z_r) - z.\epsilon(y_q)}{y.(y + \epsilon(y_q))}\right)$$

Note

Problem reduced to a system of constraints made of linear relations between integer elements only

Outiline

1. Preliminary
2. Forward & Backward Static Analysis
3. Groundwork on Constraints
4. Preliminary Results
5. Future Studies

Before Constraint Generation....

- Preliminary range measure by static analysis (no overflow)
- The accuracy may $\swarrow$ in forward analysis $\rightsquigarrow$ Weaken the pre-conditions
- The accuracy may $\nearrow$ in backward analysis $\rightsquigarrow$ Strengthen the post-conditions

$$z = x \odot y \quad with \quad \odot \in \{+, -, \times, /\}$$

$$\text{lower}(Acc_B(z)) = \begin{cases} \text{lower } Acc_B(x) \text{ in order to lower } Acc_B(z) \\ \text{lower } Acc_B(y) \text{ in order to lower } Acc_B(z) \\ \text{lower both } Acc_B(x) \text{ and } Acc_B(y) \end{cases}$$

Systematic Constraint Generation

Expression : e : := c♯ $\mathbf{p}^{\ell}$ | id^{ℓ} | $e_1^{\ell_1} +^{\ell} e_2^{\ell_2}$ | $e_1^{\ell_1} -^{\ell} e_2^{\ell_2}$ | $e_1^{\ell_1} \times^{\ell} e_2^{\ell_2}$ | $e_1^{\ell_1} \div^{\ell} e_2^{\ell_2}$
Boolean : b : := true | false | $e_1^{\ell_1} <^{\ell} e_2^{\ell_2}$ | $e_1^{\ell_1} >^{\ell} e_2^{\ell_2}$| $e_1^{\ell_1} =^{\ell} e_2^{\ell_2}$
Statement : $c ::= c_1^{\ell_1}; c_2^{\ell_2}$| $id =^{\ell} e^{\ell_1}$ | ***while***$^{\ell}$ b^{ℓ_0} ***do*** $c_1^{\ell_1}$ | ***if***$^{\ell}$ b^{ℓ_0} ***then*** $c_1^{\ell_1}$ ***else*** c |
require_accuracy(x,n)$^{\ell}$

- $l \in Lab$ unique label is attached to each expression and statement
- $\Lambda : Id \to Id \times Lab$, $x =^{l} e^{\ell_1}$
- We assign to each label l three variables : $acc_B(l), acc_F(l)$ and $acc(l)$

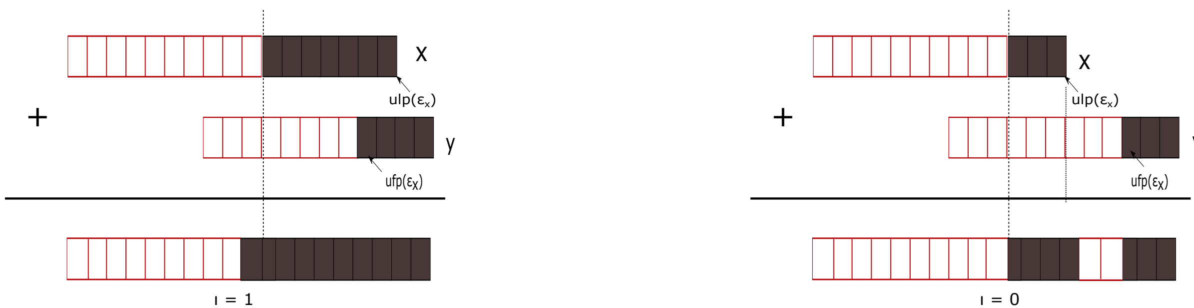
$$0 \leq acc_B(l) \leq acc(l) \leq acc_F(l)$$

Case of the Forward Addition (1/2)

$$a = ufp(x) \quad b = ufp(y)$$

$$\epsilon(x) \leq 2^{a-p+1} \quad \epsilon(y) \leq 2^{b-p+1} \quad \epsilon_+ < 2^{a-p+1} + 2^{b-p+1}$$

Definition :



$$\iota(\epsilon(x), \epsilon(y)) = \begin{cases} 0 & \text{if } ulp(\epsilon(x)) > ufp(\epsilon(y)) \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 1 :

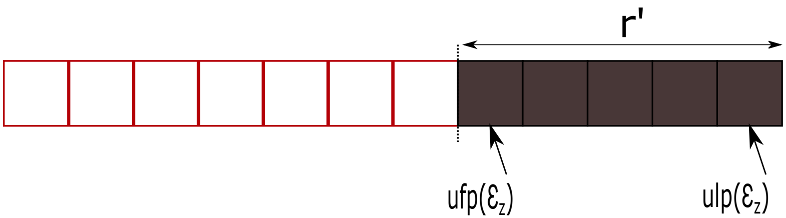
$$ufp(\epsilon_+) \leq max(a-p, b-q) + \iota(a-p, b-q)$$

$$\boxed{r_+ = ufp(x+y) - max(a-p, b-q) - \iota(a-p, b-q)}$$

Case of the Forward Addition (2/2)

$$A = ufp(\epsilon_x) \quad B = ufp(\epsilon_y) \quad C = ufp(\epsilon_z)$$

- **How to compute** $r' = ufp(\epsilon(z)) - ulp(\epsilon(z))$ **?**



- We have :

$$U = ufp(\epsilon_z) \quad and \quad u = ulp(\epsilon_z)$$

$$U = ufp(z) - R$$

$$u = min \begin{cases} ufp(x) - p - p' + 1 \\ ufp(y) - q - q' + 1. \end{cases}$$

Case of the Forward Multiplication

$$\epsilon(x) \leq 2^{a-p+1}, \epsilon(y) \leq 2^{b-p+1}$$

$$ufp(\epsilon_\times) \leq 2^{a+1}.2^{b-q+1} + 2^{b+1}.2^{a-p+1} + 2^{a-p+1}.2^{b-q+1}$$
$$= 2^{a+b-q+2} + 2^{a+b-p+2} + 2^{a+b-p-q+2}$$

$$ufp(\epsilon_\times) \leq max(a+b-p+2, a+b-q+2) + \iota(p,q)$$
$$\leq max(a+b-p+1, a+b-q+1) + \iota(p,q)$$

Thus :

$$r_\times = ufp(x \times y) - max(a+b-p+1, a+b-q+1) - \iota(p,q)$$

Linear constraints !

Outline

Syntax of IMP

Expression : **e** : := constant | id | e + e | e - e | e $\times$ e | e $\div$ e

Boolean : **b** : := true | false | e < e | e > e | e $\leq$ e | e $\geq$ e | e = e

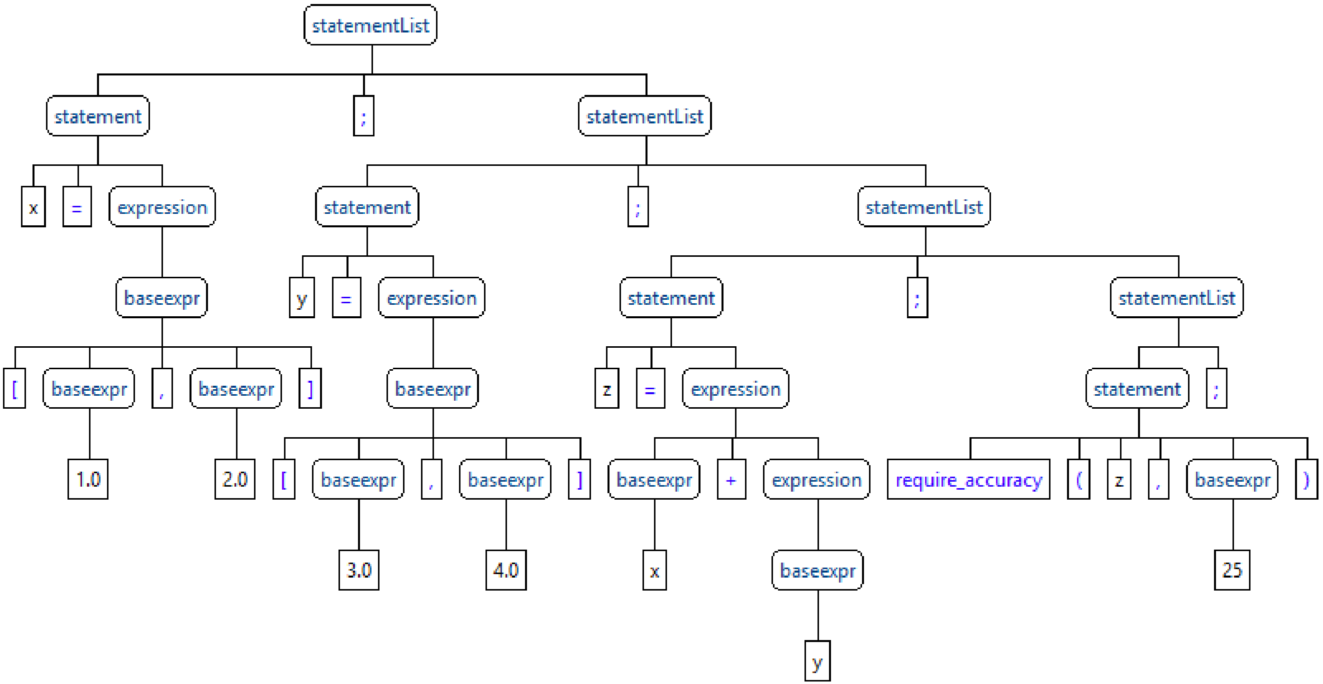
Statement : **c** : := c ; c | id = e | **while** b **do** c | **if** b **then** c **else** c

- Work Environment :
  - Java SE Development Kit 8
  - Eclipse IDE Java Oxygen.2 Release (4.7.2)
  - ANTLR4 IDE Eclipse Plugin for ANTLR 4 [1]

1. https ://github.com/antlr

Example (1/4) : Parsing Tree

```
x =[1.0, 2.0];
y =[3.0, 4.0];
z = x + y;
require_accuracy(z, 25);
```



Example (2/4) :Constraints Semantic

```
x#1# = [1.0,2.0]#0#;
y#3# = [3.0,4.0]#2#;
z#7# = Variable(x)#4# +#6# Variable(y)#5#;
require_accuracy(z,25)#9#;
```

$$\varepsilon[c_{53}^{l_0}]\Lambda = \{acc_F(l_0) = 53\}$$

$$\varepsilon[c_{53}^{l_2}]\Lambda = \{acc_F(l_2) = 53\}$$

$$\varepsilon[x^{l_4} +^{l_6} y^{l_5}]\Lambda = C[x^{l_4}]\Lambda \cup C[y^{l_5}]\Lambda \cup F_+(l_4, l_5, l_6) \cup B_+(l_4, l_5, l_6)$$

$$C[z :=^{l_7} x + y^{l_6}]\Lambda = (C, \Lambda[z \to z^{l_7}])$$

$$C[require\_accuracy(z, 25)^{l_9}]\Lambda = \{acc_B(\Lambda(z)) = 25)\}$$

Example (3/4) : Constraints Generation (Program.z3)

```
(assert (<= acc_y_lab3 accf_y_lab3))
(assert (= accf_lab0 53))
(assert (= accf_x_lab1 accf_lab0))
(assert (= accf_lab2 53))
(assert (= accf_y_lab3 accf_lab2))
(assert (= accf_lab4 53))
(assert (= accf_lab5 53))
(assert (= rSup_lab6 (- 2 (ite (> (- 1 accf_lab4) (- 2 accf_lab5)) (- 1 accf_lab4) (- 2 accf_lab5)))))
(assert (= rInf_lab6 (- 2 (ite (< (- 0 accf_lab4) (- 1 accf_lab5)) (- 0 accf_lab4) (- 1 accf_lab5)))))
(assert (= ioSup_lab6 (ite(> 2 1) 0 1)))
(assert (= propaUlpSup_lab6 (ite (< ulpL1Sup_lab4 ulpL2Sup_lab5) ulpL1Sup_lab4 ulpL2Sup_lab5)))
(assert (= ioInf_lab6 (ite(> 0 1) 0 1)))
(assert (= accf_lab6 (ite (< (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6)) (- rInf_lab6 ioInf_lab6) (- rSup_
(assert (= accf_z_lab7 accf_lab6))
(assert (= accb_x_lab1 accb_lab0))
(assert (= accb_y_lab3 accb_lab2))
(assert (= rSup_lab6 (- 2 (ite (> (- 1 accf_lab4) (- 2 accf_lab5)) (- 1 accf_lab4) (- 2 accf_lab5)))))
(assert (= rInf_lab6 (- 2 (ite (< (- 0 accf_lab4) (- 1 accf_lab5)) (- 0 accf_lab4) (- 1 accf_lab5)))))
(assert (= ioSup_lab6 (ite(> 2 1) 0 1)))
(assert (= propaUlpSup_lab6 (ite (< ulpL1Sup_lab4 ulpL2Sup_lab5) ulpL1Sup_lab4 ulpL2Sup_lab5)))
(assert (= ioInf_lab6 (ite(> 0 1) 0 1)))
(assert (= accf_lab6 (ite (< (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6)) (- rInf_lab6 ioInf_lab6) (- rSup_
(assert (= slSup_lab5 (- 1 (- 2 accb_lab6))))
(assert (= slInf_lab5 (- 0 (- 2 accb_lab6))))
(assert (= accb_lab5 (ite (> slInf_lab5 slSup_lab5) slInf_lab5 slSup_lab5)))
(assert (= slSup_lab4 (- 2 (- 2 accb_lab6))))
(assert (= slInf_lab4 (- 1 (- 2 accb_lab6))))
(assert (= accb_lab4 (ite (> slInf_lab4 slSup_lab4) slInf_lab4 slSup_lab4)))
(assert (or (and(<= acc_lab4 accf_lab4) (>= acc_lab5 accb_lab5)) (and(<= acc_lab5 accf_lab5) (>= acc_lab4 accb_la
(assert (= accb_z_lab7 accb_lab6))
(assert (= accb_z_lab9 25))
```

Example (4/4) : Z3 SMT solver solution

$x^{|25|} = [1.0, 2.0]^{|25|};$
$y^{|24|} = [3.0, 4.0]^{|24|};$
$z^{|25|} = [1.0, 2.0]^{|25|} +^{|25|} [3.0, 4.0]^{|24|};$
require_accuracy(z,25);

Number_Variables= 49
Number_Constraints = 57

Cost function

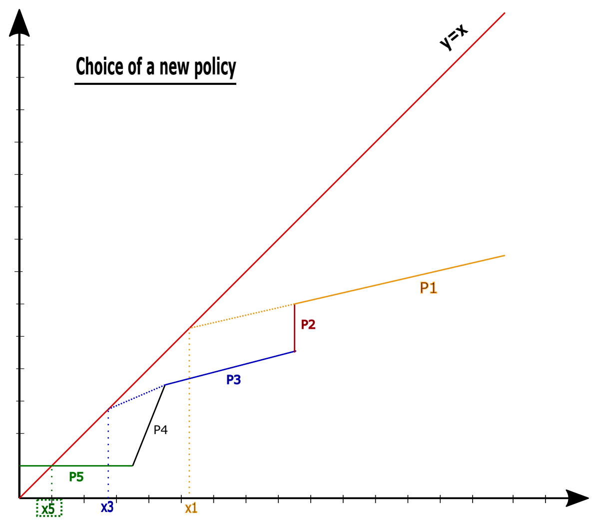
Solutions are not unique. We need to add an additional constraint related to a cost function ϕ to the constraints

$$\phi(c) = \sum_{x \in Id, l \in Lab} acc(x^l) + \sum_{l \in Lab} acc(l)$$

Outline

Policy Iteration

- **Motivation**
  - Z3 SMT solver = decision tool $\neq$ optimization tool
- **Idea**
  - Using **Policy iteration** to improve accuracy [1]
  - Generated constraints are of the form **min-max of discrete affine maps**
  - Feeding the policy iteration with the z3 solution as an initial policy !
- **Finality**
  - Comparing the policy iteration and Z3 solutions (in term of execution time and optimality)



Conclusion

- Floating-point computations determination minimal precision
- **Contribution**
  - Forward & Backward static analysis for numerical accuracy
  - Formulation as first order linear constraints
- **Extensions :** functions, arrays, fixed-point arithmetic, etc.
- **Minimal precision determination :** Policy Iteration
- **Experimentally tool validation :** embedded systems, numerical computation, etc.

References

Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou.
Static analysis by policy iteration on relational domains.
In Rocco De Nicola, editor, *Programming Languages and Systems*, pages 237–252, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
32

Stef Graillat, Fabienne Jézéquel, Romain Picot, François Févotte, and Bruno Lathuilière.
PROMISE : floating-point precision tuning with stochastic arithmetic.
In *17th international symposium on Scientific Computing, Computer Arithmetic and Verified Numerics (SCAN 2016)*, pages 98–99, UPPSALA, Sweden, September 2016.
5

Matthieu Martel.
Floating-point format inference in mixed-precision.
In *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*.
5

THANK YOU FOR LISTENING

Q & A !