AutoSense : A Framework for Automated Sensitivity Analysis of Program Data

Bernard Nongpoh[1], Rajarshi Ray[1], Saikat Dutta [2], Ansuman Banerjee [3]

[1] Department of Computer Science and Engineering, National Institute of Technology Meghalaya, Shillong, India.
[2] Department of Computer Science and Engineering, Jadavpur University, Kolkata, India.
[3] Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India.

11th European Symposium on Software Engineering and Foundations of Software Engineering (ESEC-FSE), Paderborn, Germany.
September 6, 2017

OUTLINE

Sensitivity Analysis: A Motivating Example

```
bool binsearch(int lo, int hi)
{
    unsigned int size = hi-lo + 1;
    unsigned int mid = (lo+hi)/2;
    if(lo>hi) return false;
    if (size >= 1){
        if(a[mid] == key) return true;
        else if(a[mid]>key)
            return binsearch(lo, mid-1);
        else return binsearch(mid+1, hi);
    }
    return false;
}
```

Sensitive **Insensitive**

Note that any erroneous value except 0 that size may take, the program behavior remains unaffected.

Problem Statement and Contributions

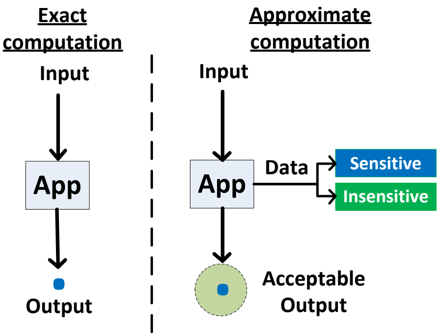


Figure 1: Data classification in approximate computing

Contributions:

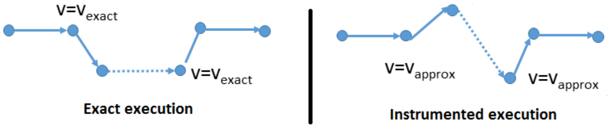
- A Dynamic analysis to automatically classify program data as sensitive or insensitive with probabilistic guarantee.
- A Static-Dynamic combined analysis for efficiency.

Definition: Sensitive Data

Given an acceptable QoS band for a program $\mathcal{P}$ and a sensitivity threshold probability θ, a program data $v \in \mathcal{D}$ is called sensitive if and only if $\forall e \in E$, the probability that the program output remains in the acceptable QoS band when every instance (v_e, ℓ) in e is replaced with some (v_{approx}, ℓ), is less than θ.

$$\mathcal{SD} = \{ v \in \mathcal{D} \mid \forall e \in E, \forall \ell \in \ell_v^e, (v_e, \ell) \rightarrow (v_{approx}, \ell) \implies Pr(\mathcal{R} \in QoS\ ) < \theta \} \tag{1}$$

where $(v_e, \ell) \rightarrow (v_{approx}, \ell)$ means the substitution of (v_{approx}, ℓ) in place of (v_e, ℓ). The set of *insensitive* data is defined as $\overline{\mathcal{SD}} = \mathcal{D} - \mathcal{SD}$.



Dynamic Sensitivity Analysis

For every $v \in \mathcal{D}$, we propose a hypothesis that $\forall e \in E, \forall \ell \in \ell_v^e, (v_e, \ell) \rightarrow (v_{approx}, \ell) \implies \mathcal{R} \in QoS$. Let us denote such an hypothesis by K. Test the following null and contrary hypothesis:

$$H : Pr(K) < \theta; H' : Pr(K) \geq \theta \tag{2}$$

where $Pr(K)$ is the probability that the hypothesis K is true.

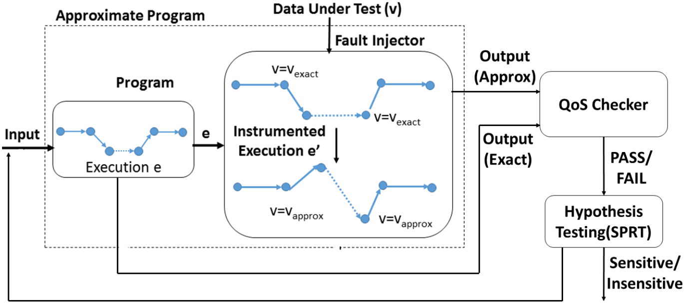


Figure 2: Framework of Dynamic Sensitivity Analysis with Hypothesis Testing

Static-Dynamic combined Sensitivity Analysis

Limitation of Dynamic Sensitivity Analysis

- Compute and data intensive programs may take a long time to terminate, making each trial during the hypothesis testing expensive.
- Generating random inputs for many applications can be challenging.

Static-Dynamic combined analysis

- Based on classical data flow analysis.
- The sensitivity of some variables like global data, method parameters and method local data whose expression has constant(s) or function call(s) is initialized using dynamic analysis.
- The principle behind our static analysis is that insensitive data cannot flow into sensitive data.

Static-Dynamic Combined Sensitivity Analysis

Definition: Data Sensitivity Lattice

The elements of the complete lattice L of our analysis are mappings $\sigma : \mathcal{D} \to \{\bot, S, I, \top\}$.

- $\sigma(x) = \bot$ denotes that no information is known about the data x
- $\sigma(x) = \top$ denotes that x may be *sensitive* or *insensitive*.
- $\sigma(x) = S$ and $\sigma(x) = I$ denotes x to be *sensitive* and *insensitive* respectively.

We define a *data sensitivity lattice* over the range of σ, i.e., $\{\bot, S, I, \top\}$

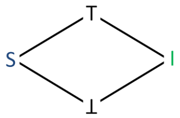


Figure 3: Data Sensitivity Lattice

Static-Dynamic Combined Sensitivity Analysis

Partial order on σ

The partial order on σ is defined :

$$\begin{aligned} &\forall\sigma : \bot_\sigma \sqsubseteq \sigma \\ &\forall\sigma_1, \sigma_2 : \sigma_1 \sqsubseteq \sigma_2 \textit{ iff } \forall x, \sigma_1(x) \sqsubseteq_D \sigma_2(x). \end{aligned} \tag{3}$$

where $\bot_\sigma \in \sigma$ maps every $x \in \mathcal{D}$ to $\bot$, $\sqsubseteq$ denotes the partial order relation on σ and $\sqsubseteq_D$ denotes the partial order relation of the *data sensitivity lattice*.

The *join* operation over σ is defined in Eq. 4.

$$(\sigma_1 \sqcup \sigma_2)(x) = \sigma_1(x) \sqcup_D \sigma_2(x) \tag{4}$$

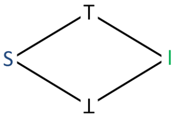


Figure 4: Data Sensitivity Lattice

Static-Dynamic Combined Sensitivity Analysis

Transfer Functions

Considering a general assignment statement block $[x := a]$, a being any expression, we define the transfer functions of our analysis as:

$$[x = a] : f(\sigma) = \begin{cases} \sigma(x \to I) & \textit{if } \forall v \in FV(a), \sigma(v) = I \\ \sigma(x \to S) & \textit{if } \forall v \in FV(a), \sigma(v) = S \\ \sigma(x \to \top) & \textit{if } \exists u,\ v \in FV(a) \\ & \quad \textit{s.t. } \sigma(u) = S,\ \sigma(v) = I \\ \sigma & \textit{if } FV(a) = \emptyset \end{cases} \tag{5}$$

$$[\cdots] : f(\sigma) = \sigma$$

where $[\cdots]$ is to denote any program statement which is not an assignment statement and $FV(a)$ is the set of all free variables of the expression a.

Reliability of Sensitivity Analysis

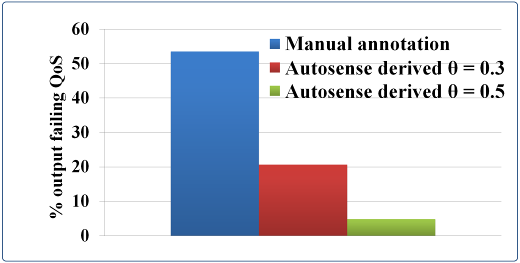
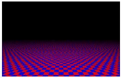
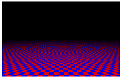


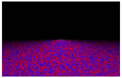
Figure 5: Percent output failing QoS with confidence $\theta = 0.3$ and $\theta = 0.5$



(a)
Original image



(b)
AutoSense guided approximation



(c)
Manual guided approximation

Figure 6: Raytracer rendered image with AutoSense guided approximation

Evaluation of Dynamic Analysis

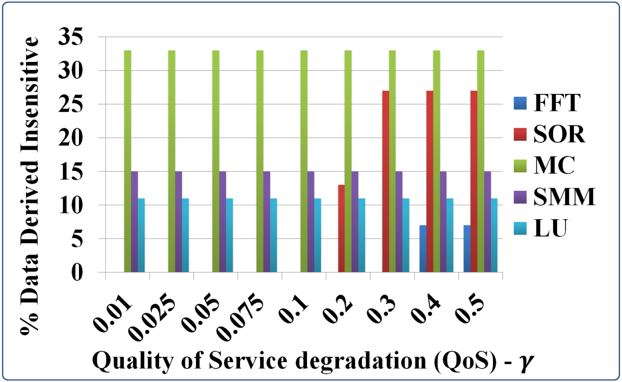


Figure 7: The percent insensitive data reported by a AutoSense on varying QoS γ and fixed probability factor $\theta = 0.5$

Evaluation of Dynamic Analysis

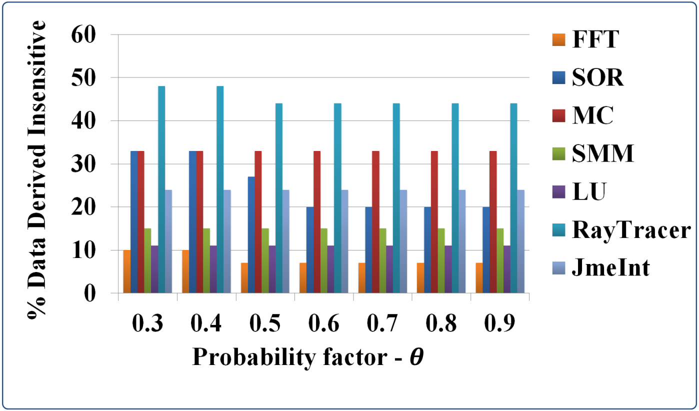


Figure 8: The percent insensitive data reported by a AutoSense on varying θ and fixed QoS $\gamma = 0.5$ (scimark2), PSNR=10.5 (raytracer) and exact (jmeint)

Evaluation of Static-Dynamic Combined Analysis

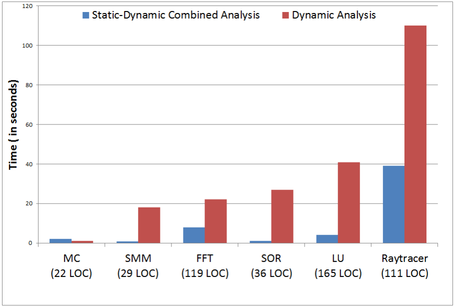


Figure 9: Performance of Static-dynamic combined vs. Dynamic analysis

Application	TP	FP	FN	Precision (%)	Recall (%)
FFT	0	0	3	0	0
SOR	3	0	0	100	100
MC	1	0	1	100	50
SMM	2	0	0	100	100
LU	0	0	9	0	0
Raytracer	0	1	2	0	0

Table 1: Precision, Recall of the Combined Analysis w.r.t. Dynamic Analysis

Conclusion

- Identification of insensitive error resilient data of an application is non-trivial, especially when the application is large and has substantial data and control dependencies.
- We Illustrate that a systematic study of the effect of inaccuracy in program data with statistical methods like hypothesis testing can lead to automatic classification of insensitive and sensitive data.
- Dynamic analysis is computationally expensive and time consuming for many applications. We propose a static analysis to derive insensitive data, with efficiency. Although static analysis shows high precision w.r.t. dynamic analysis, it fails to identify many.

Thank You ...