

# Embedded Systems



**Arijit Mondal**

Dept. of Computer Science & Engineering  
Indian Institute of Technology Patna

[arijit@iitp.ac.in](mailto:arijit@iitp.ac.in)

# Quantitative Analysis

# Introduction

- Quantitative properties
  - Position, velocity, temperature, weight, reaction time, etc.
- Execution time analysis
- Power analysis
- Need to analyze program / software
- Require models of platform

# Target

- **Generic problem**

- A task defined by a program  $P$ , environment  $E$  and quantity of interest  $q$ , then  $q = f_p(x, w)$

- $x$  - input to program
- $w$  - environment parameters

- In most of the cases extreme value of the quantity is of interest

- Extreme case analysis:  $\max_{x,w} f_p(x, w)$  or  $\min_{x,w} f_p(x, w)$

- It may be difficult to determine exact value

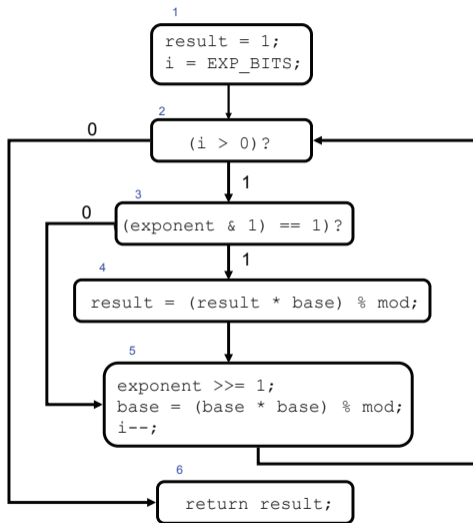
- Threshold analysis:  $\forall x, w f_p(x, w) \leq T$  or  $\forall x, w f_p(x, w) \geq T$

- Average case analysis:  $E_{D_x, D_w} f_p(x, w)$

# Modular exponentiation: $b^e \bmod n$

```
#define EXP_BITS 32
typedef unsigned int UI;
UI modexp(UI base, UI exponent, UI mod) {
    int i;
    UI result = 1;
    i = EXP_BITS;
    while(i > 0) {
        if ((exponent & 1) == 1) {
            result = (result * base) % mod;
        }
        exponent >>= 1;
        base = (base * base) % mod;
        i--;
    }
    return result;
}
```

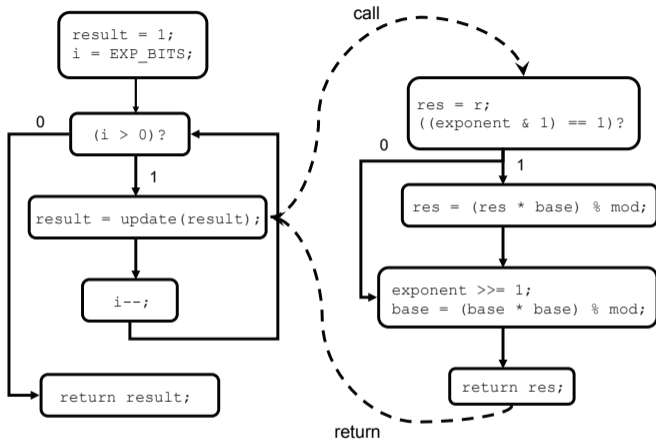
# Control flow diagram



# Modular exponentiation: $b^e \bmod n$

```
#define EXP_BITS 32
typedef unsigned int UI;
UI exponent, base, mod;
UI update(UI r) {
    UI res = r;
    if ((exponent & 1) == 1) { res = (res * base) % mod; }
    exponent >>= 1;
    base = (base * base) % mod;
    return res;
}
UI modexp_call() {
    UI result = 1; int i;
    i = EXP_BITS;
    while(i > 0) { result = update(result); i--; }
    return result;
}
```

# Control flow diagram





# Execution time: Loop bounds

```
for(i=EXP_BITS; i > 0; i--) {  
    if ((exponent & 1) == 1) {  
        result = (result * base) % mod;  
    }  
    exponent >>= 1;  
    base = (base * base) % mod;  
}
```

```
while (exponent != 0) {  
    if ((exponent & 1) == 1) {  
        result = (result * base) % mod;  
    }  
    exponent >>= 1;  
    base = (base * base) % mod;  
}
```

# Execution time: Exponential path space

```
void count() {  
    int Outer, Inner;  
    for (Outer = 0; Outer < MAXSIZE; Outer++) {  
        for (Inner = 0; Inner < MAXSIZE; Inner++){  
            if (Array[Outer][Inner] >= 0) {  
                Ptotal += Array[Outer][Inner];  
                Pcnt++;  
            } else {  
                Ntotal += Array[Outer][Inner];  
                Ncnt++;  
            }  
        }  
    }  
}
```

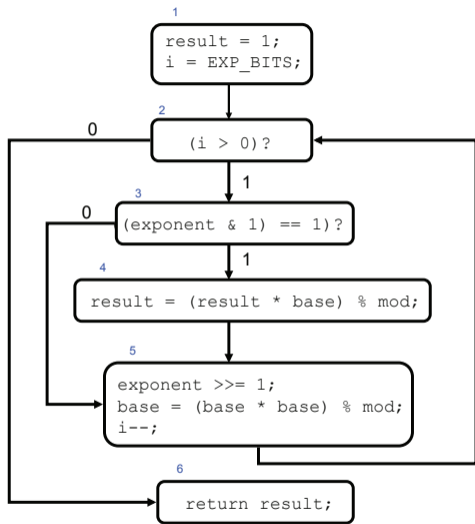
# Execution time: Path feasibility

```
void altitude_control_task(void) {
    if (pprz_mode == PPRZ_MODE_AUTO2 || pprz_mode == PPRZ_MODE_HOME) {
        if (vertical_mode == VERTICAL_MODE_AUTO_ALT) {
            float err = estimator_z - desired_altitude;
            desired_climb = pre_climb + altitude_pgain * err;
            if (desired_climb < -CLIMB_MAX) {
                desired_climb = -CLIMB_MAX;
            }
            if (desired_climb > CLIMB_MAX) {
                desired_climb = CLIMB_MAX;
            }
        }
    }
}
```

# Execution time: Memory hierarchy

```
float dot_product(float *x, float *y, int n) {  
    float result = 0.0;  
    int i;  
    for(i=0; i < n; i++) {  
        result += x[i] * y[i];  
    }  
    return result;  
}
```

# Constraints for CFG



$$X_1 = 1$$

$$X_6 = 1$$

$$X_1 = d_{12}$$

$$X_2 = d_{12} + d_{52} = d_{23} + d_{26}$$

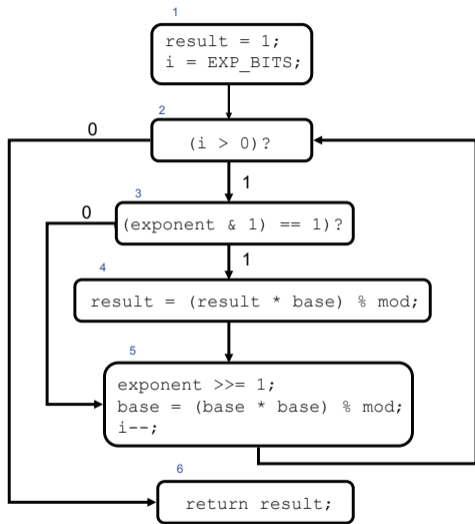
$$X_3 = d_{23} = d_{34} + d_{35}$$

$$X_4 = d_{34} = d_{45}$$

$$X_5 = d_{35} + d_{45} = d_{52}$$

$$X_6 = d_{26}$$

# Constraints for CFG



$$x_1 = 1$$

$$x_6 = 1$$

$$x_1 = d_{12}$$

$$x_2 = d_{12} + d_{52} = d_{23} + d_{26}$$

$$x_3 = d_{23} = d_{34} + d_{35}$$

$$x_4 = d_{34} = d_{45}$$

$$x_5 = d_{35} + d_{45} = d_{52}$$

$$x_6 = d_{26}$$

$$\max_{x_i, 1 \leq i \leq n} \sum_i w_i x_i$$