Universidade do Minho
Escola de Engenharia

HASLab
INESCTEC

Green
Software
Lab

STATIC ENERGY PREDICTION IN SOFTWARE:

A WORST-CASE SCENARIO APPROACH

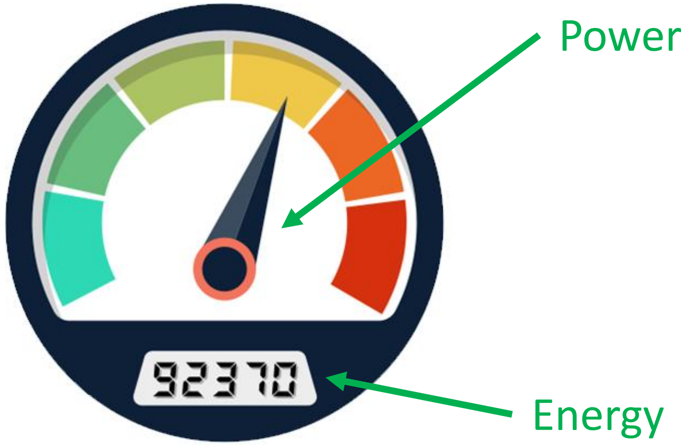
Marco Couto

marco.l.couto@inesctec.pt

MOTIVATION

ENERGY VS. POWER

- *Power (w)* – rate (or effort) at which that work is done
- *Energy* (J) – amount of work done
- *Power* can change constantly while *Energy* is the accumulation



Energy = Power x Seconds

360,000 J = 100W x 3,600s

ENERGY EFFICIENT SOFTWARE

- Programmers problems:
  - How to analyze
  - How to interpret
  - How to improve
- Researchers problem:
  - How to provide information to developers

Mining questions about software energy consumption - [MSR’14]

Integrated energy-directed test suite optimization - [ISTA’14]

Seeds: A software engineer’s energy-optimization decision support framework - [ICSE’14]

Green Software Lab

THE CHALLENGE

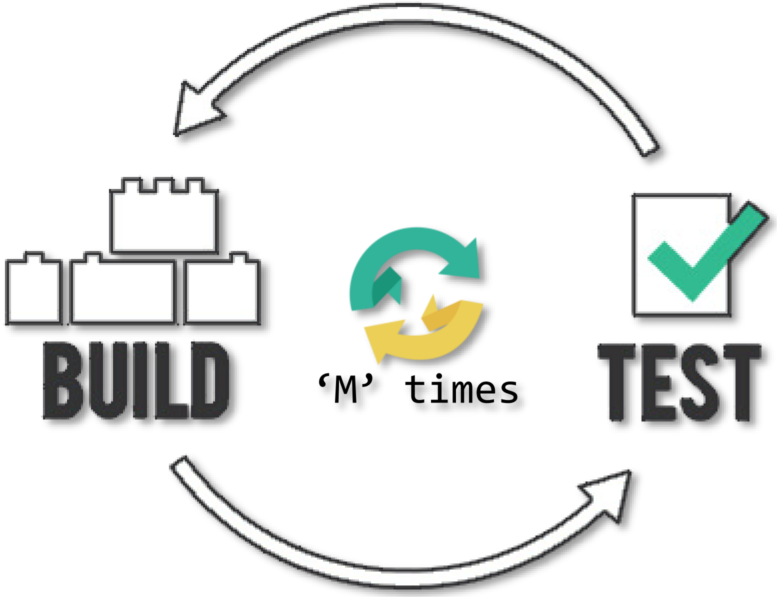
STATIC ENERGY ANALYSIS

WHY?

- Dynamic can be too costly
  1. Build application + tests
  2. Intrument app and/or tests
  3. Run tests
  4. Gather measurements and analyze them
  5. Repeat N times

- ‘M’ apps -> Repeat all ‘M’ times

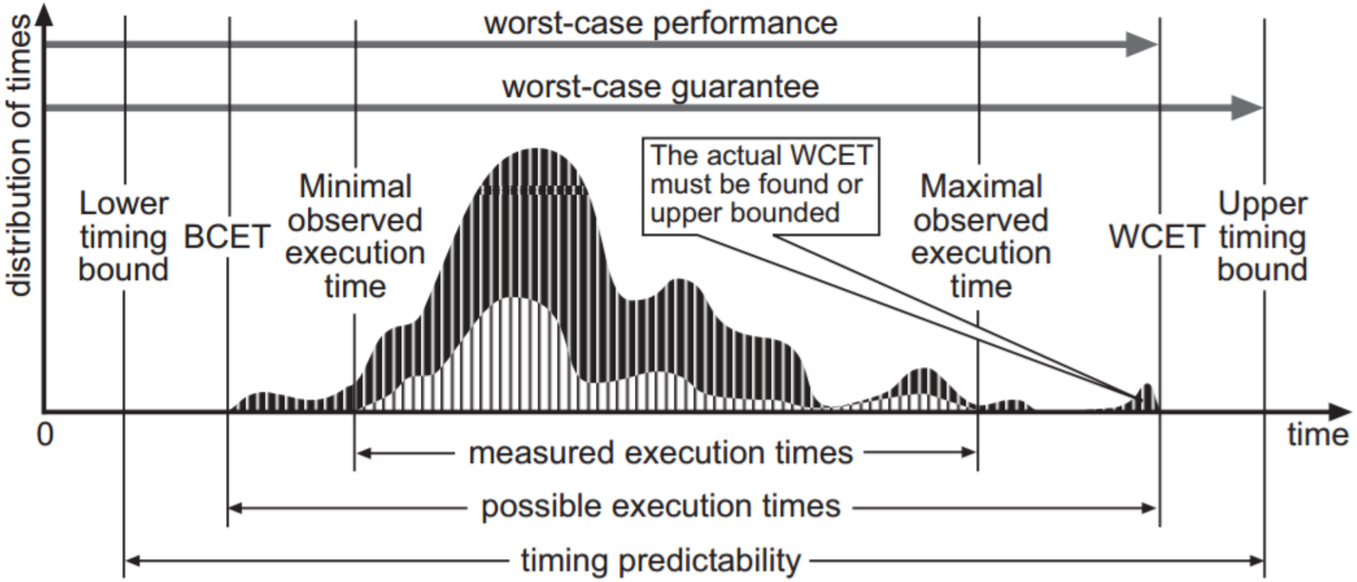
- Are all cases covered?



STATIC ENERGY ANALYSIS

HOW?

- How is it done for execution time?
  - WCET – Worst Case Execution Time

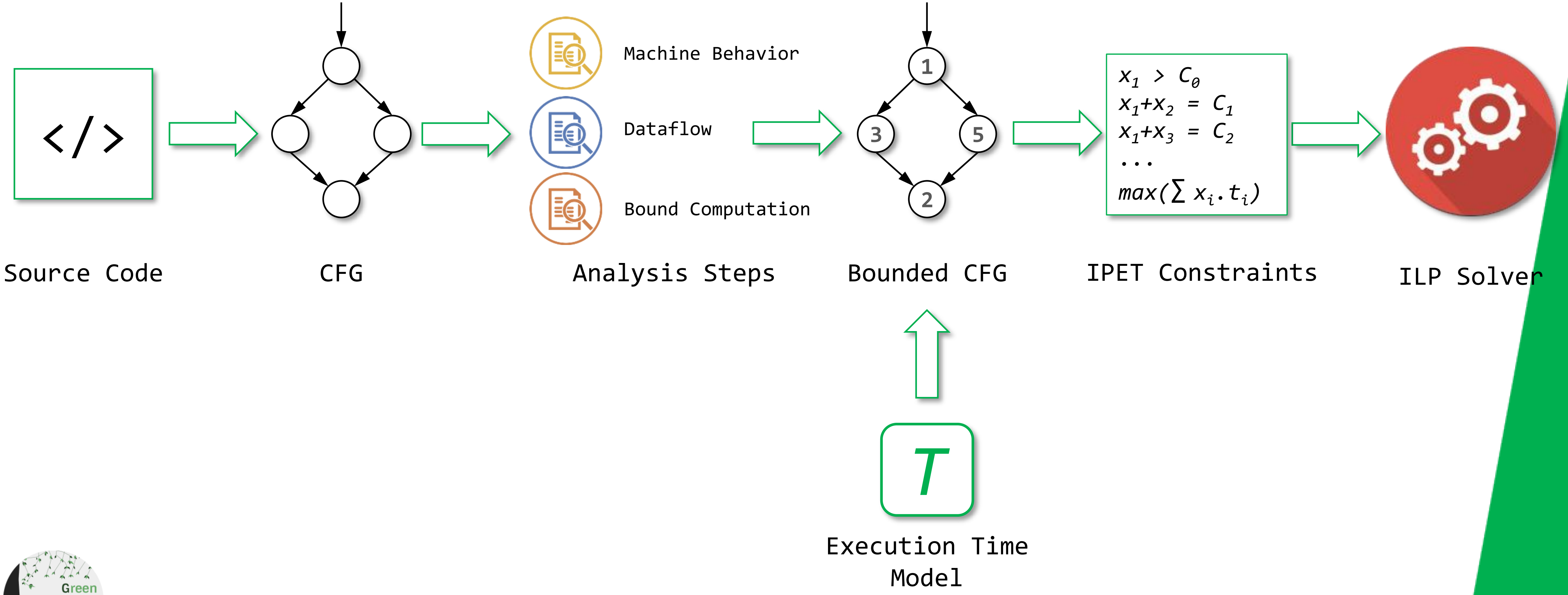


- Can we do the same for energy?

STATIC ~~ENERGY~~ EXECUTION TIME ANALYSIS

HOW?

How exactly does WCET works?



</>

Source Code

CFG

Machine Behavior

Dataflow

Bound Computation

Analysis Steps

Bounded CFG

$x_1 > C_0$
$x_1+x_2 = C_1$
$x_1+x_3 = C_2$
...
$max(\sum x_i . t_i)$

IPET Constraints

ILP Solver

T

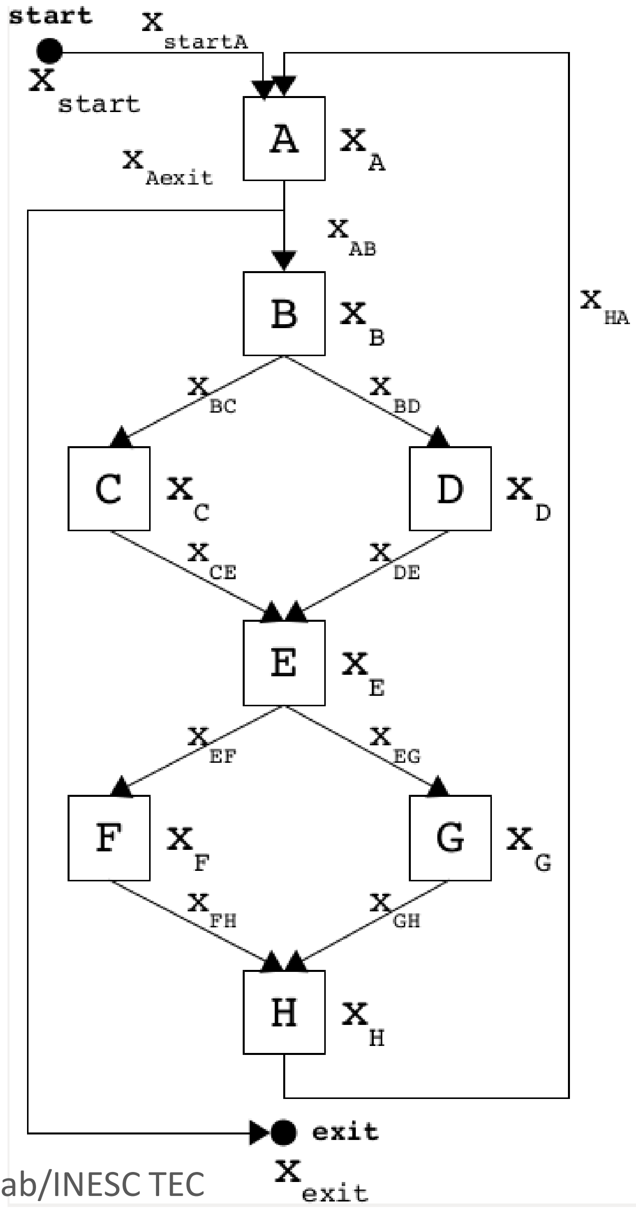
Execution Time Model

IPET – IMPLICIT PATH ENUMERATION TECHNIQUE

A.K.A CONSTRAINT GENERATOR

- Each node has:
  - A count variable, ‘*x*’
  - A cost variable, ‘*t*’
- Each edge has:
  - A count variable, ‘*x*’

- 4 Simple but strong premises:
  1. Start and end nodes accessed exactly 1 time
  2. #*x* of node i = sum (#*x* of all pred. of i)
  3. #*x* of node i = sum (#*x* of all succ. of i)
  4. Loops must have na upper bound



//start and exit conditions

$x_{start}=1,\ x_{exit}=1$

//structural conditions

$x_{start}=x_{startA}$

$x_A=x_{startA}+x_{HA}=x_{Aexit}+x_{AB}$

$x_B=x_{AB}=x_{BC}+x_{BD}$

$x_C=x_{BC}=x_{CE}$

...

$x_H=x_{FH}+x_{GH}=x_{HA}$

$x_{exit}=x_{Aexit}$

//Loopbound constraint

$x_A<=100$

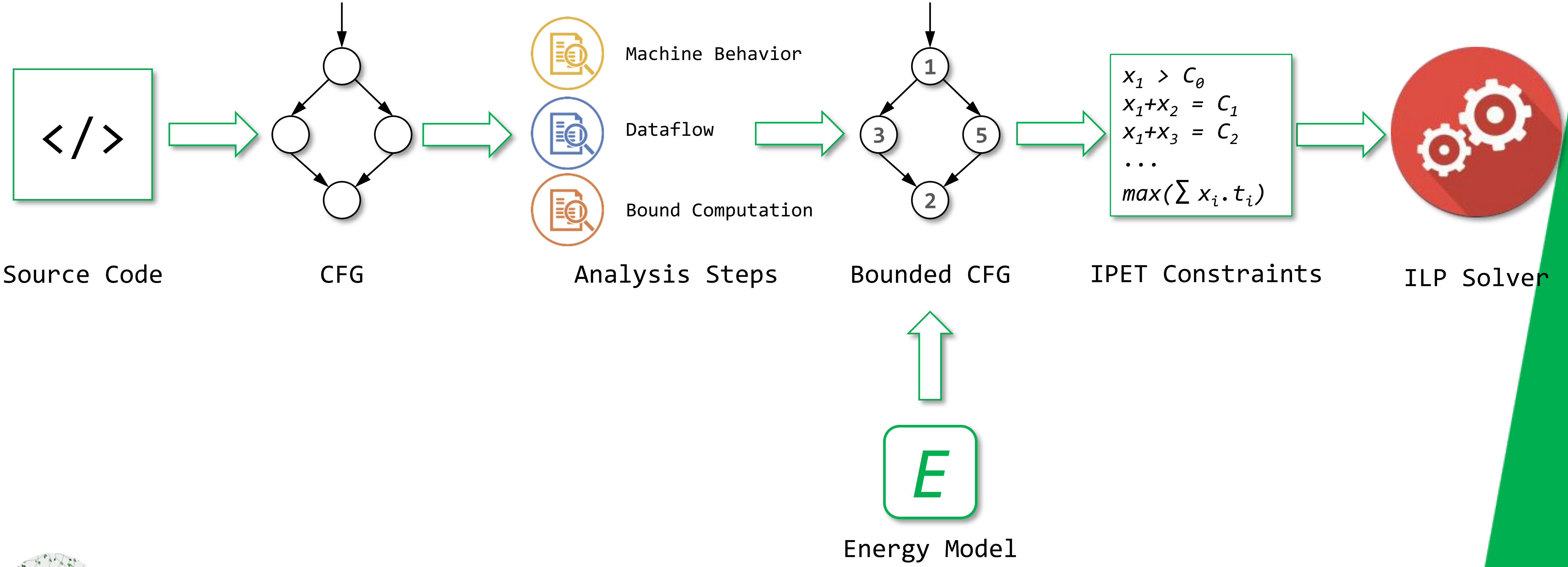
//WCET Expression

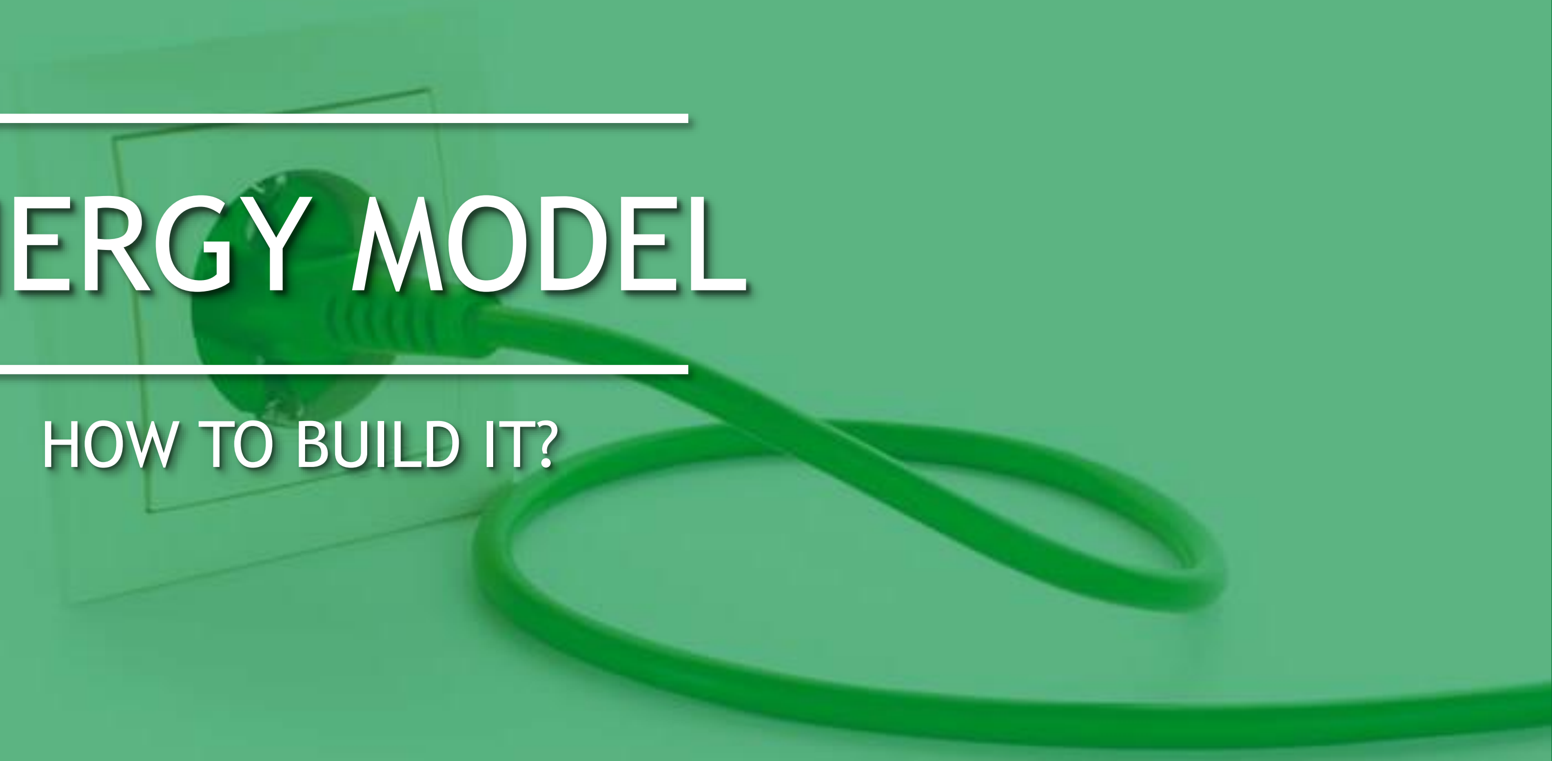
$WCET=max(x_A*t_A + x_B*t_B +...+ x_H*t_H)$

STATIC ENERGY ANALYSIS

HOW?

How exactly does WCEC works?





ENERGY MODEL

HOW TO BUILD IT?

BUILDING THE ENERGY MODEL

Estimating Mobile Application Energy Consumption using Program Analysis - [ICSE'13]

- Two types of instructions:
  - Path-Independent:
    - `x=1;`
    - `i++;`
    - `a=2*b;`
    - `return 0;`
  - Path-Dependent:
    - Allocations: `malloc(sizeof(int))` ≠ `malloc(sizeof(double))`
    - System calls: `scanf("%d", &a)` ≠ `scanf("%f", &a)`
    - Other functions: `strcmp(...)`, `atoi(...)`, `sizeof(...)`, ...
- Some instructions depend on others:
  - '`int x = z;`' depends on the declaration of `z`
  - '`printf("%d", a);`' assumes `a` has a previous assigned value

BUILDING THE ENERGY MODEL

Estimating Mobile Application Energy Consumption using Program Analysis - [ICSE'13]

- Path-Independent:
  - Execute **N** times
  - Repeat **M** times, measure each time and **sum** all
  - At the end: *`energy = (sum/M/N) – dependencies`*

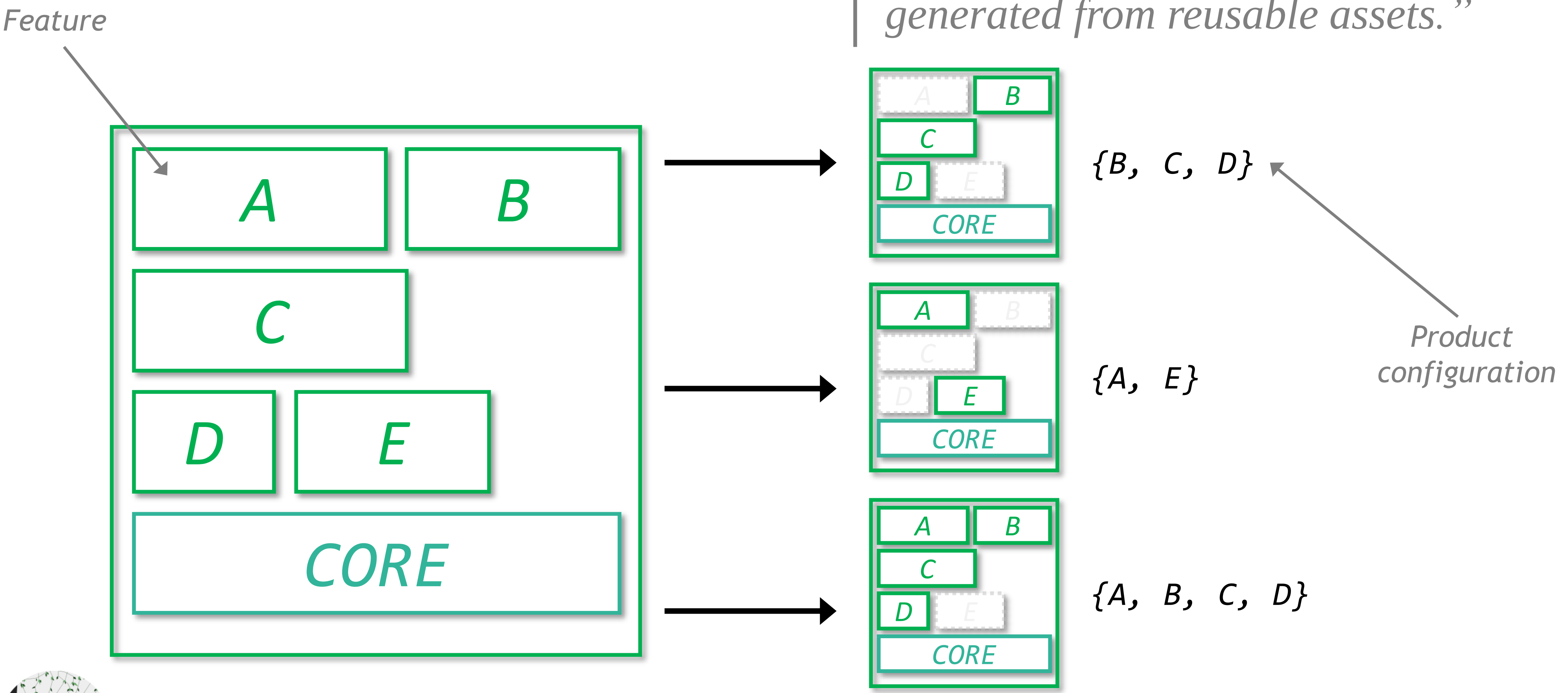
- Path-Dependent:
  - Execute **N** times
  - Repeat **M** times, measure each time
  - At the end: *`energy = max(all measures) – dependencies`*

SOFTWARE PRODUCT LINES

THE CASE STUDY

SOFTWARE PRODUCT LINE

> *"(…) a set of related software products (…) generated from reusable assets."*



Feature

`{B, C, D}`

`{A, E}`

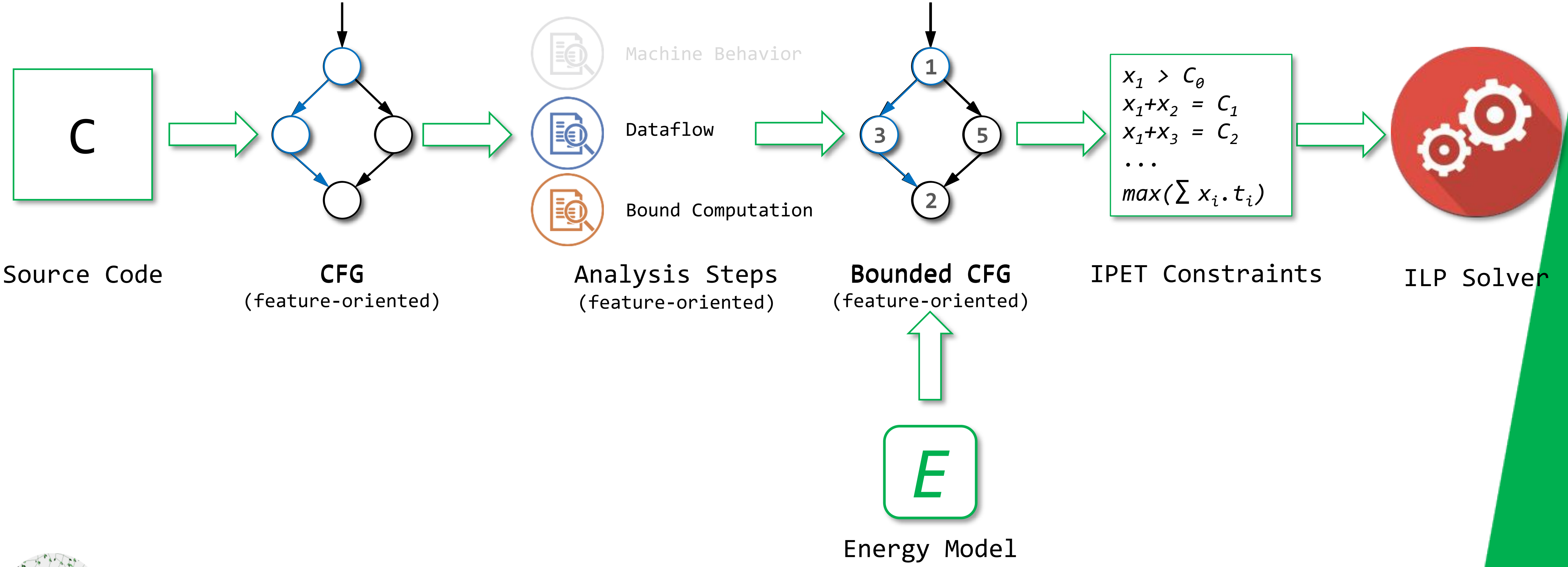
`{A, B, C, D}`

Product configuration

STATIC ENERGY ANALYSIS

THIS TIME, FOR SPL!

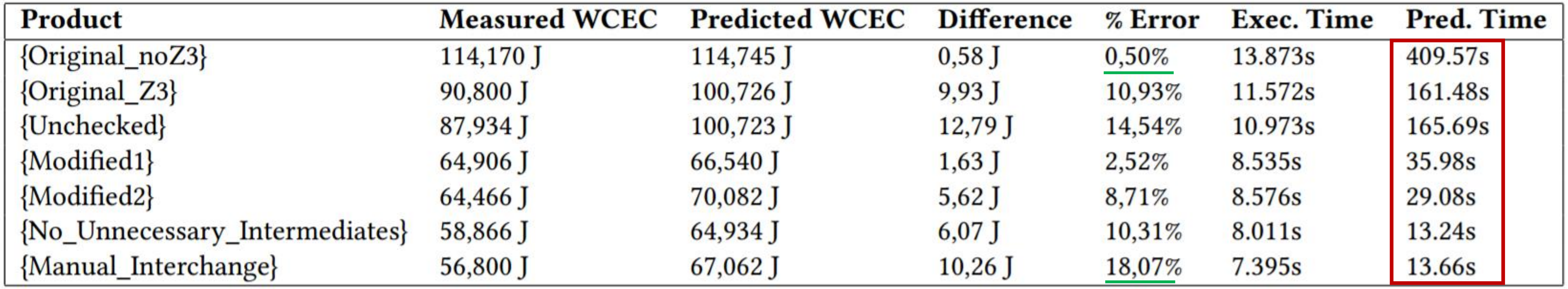
Intraprocedural Dataflow Analysis for Software Product Lines
- [AOSD '12]



RESULTS

PRELIMINARY RESULTS

- A SPL with 7 (exclusive) *features*

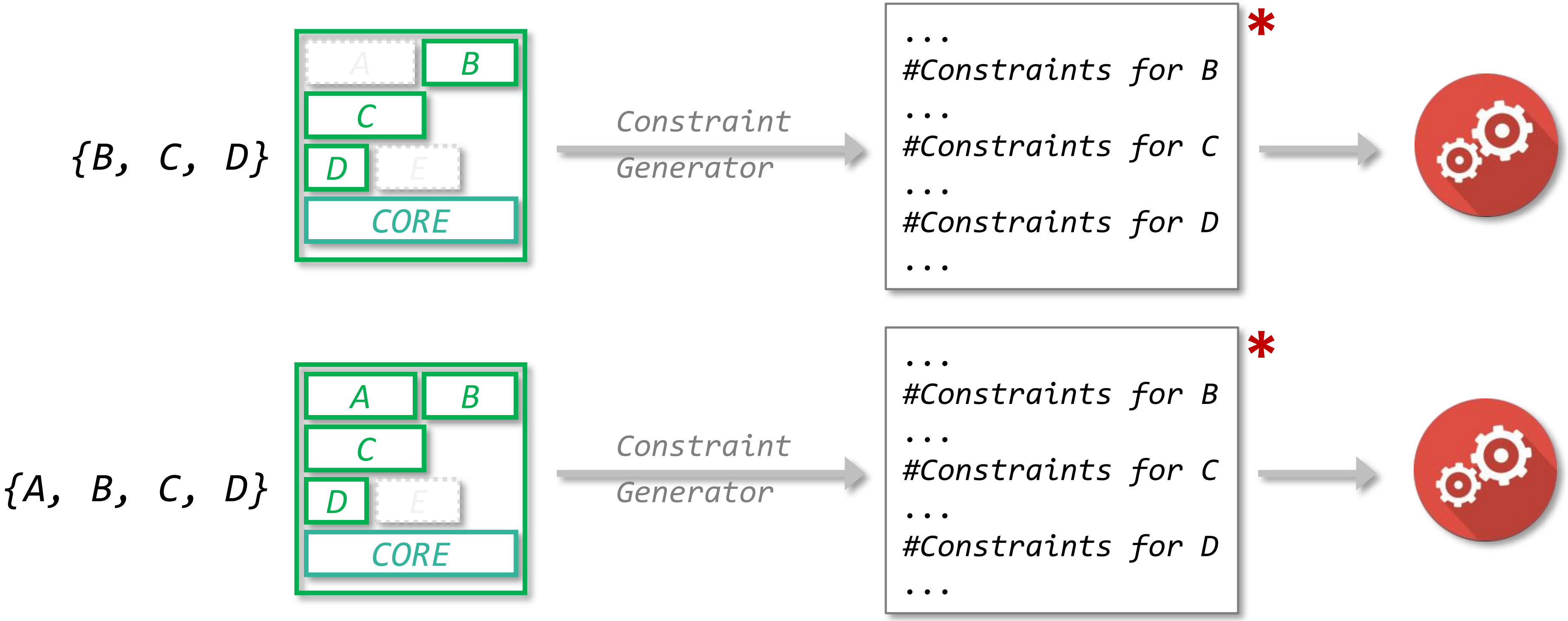


Product	Measured WCEC	Predicted WCEC	Difference	% Error	Exec. Time	Pred. Time
{Original_noZ3}	114,170 J	114,745 J	0,58 J	0,50%	13.873s	409.57s
{Original_Z3}	90,800 J	100,726 J	9,93 J	10,93%	11.572s	161.48s
{Unchecked}	87,934 J	100,723 J	12,79 J	14,54%	10.973s	165.69s
{Modified1}	64,906 J	66,540 J	1,63 J	2,52%	8.535s	35.98s
{Modified2}	64,466 J	70,082 J	5,62 J	8,71%	8.576s	29.08s
{No_Unnecessary_Intermediates}	58,866 J	64,934 J	6,07 J	10,31%	8.011s	13.24s
{Manual_Interchange}	56,800 J	67,062 J	10,26 J	18,07%	7.395s	13.66s

Results submited to SPLC'17 – Software Product Lines Conference, 21st edition:
"Products go green: Worst-Case Energy Consumption in Software Product Lines"

WHAT NOW?

- This work was focused on accuracy
- Technique's performance could be improved...
  - Main reason: shared constraints are re-calculated for every *product*



```
...
#Constraints for B
...
#Constraints for C
...
#Constraints for D
...
```

```
...
#Constraints for B
...
#Constraints for C
...
#Constraints for D
...
```

*Constraints solved more than once!

WHAT NOW?

- Accuracy can also be improved!
  - Include a machine behavior analysis
    - -> different energy consumptions per instruction
  - Consider lower abstraction level
    - For now, we analyzed *C* code

- Apply technique to different contexts
  - Mobile?
  - Embedded?
  - ???

Universidade do Minho
Escola de Engenharia

HASLab
INESCTEC

Green
Software
Lab

STATIC ENERGY PREDICTION IN SOFTWARE:

A WORST-CASE SCENARIO APPROACH

Marco Couto

marco.l.couto@inesctec.pt