A Quantitative Analysis of Errors in Linux

Elvis Flesborg
@ Helenekilde Badehotel, Tisvildeleje

April 20, 2015

About Me

- Elvis Flesborg
- Master thesis at IT University of Copenhagen
- Specialization in scalable computing
- Claus Brabrand as supervisor
- Jean Melo as co-supervisor

Objective

- Quantify errors in Linux

Objective

- Quantify errors in Linux
- In contrast to [42bugs], which is a qualitative analysis

Objective

- Quantify errors in Linux
- In contrast to [42bugs], which is a qualitative analysis
  - Linux has 14,172 different features
    - grep -r "^\ *config " | grep Kconfig | awk -F':' '{print $ 2}' | sort | uniq | wc
  - That is $\underline{2^{14,172}}$ different configurations minus invalid configurations.
  - More than estimated number of atoms in the known universe.

Research Questions

- How many bugs? (percentage)

Research Questions

- How many bugs? (percentage)
- What are the most common types of bugs?

Research Questions

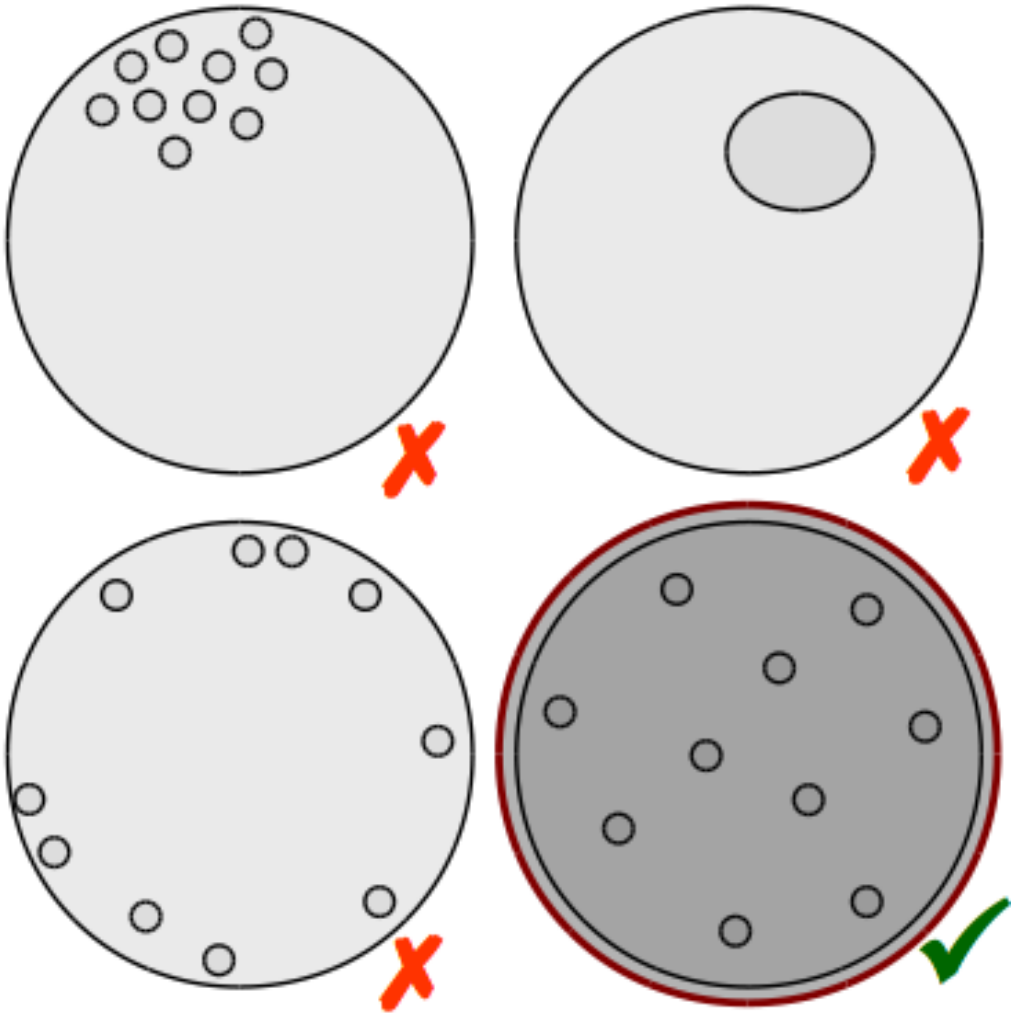
- How many bugs? (percentage)
- What are the most common types of bugs?
- Where are most bugs located?

Research Questions

- How many bugs? (percentage)
- What are the most common types of bugs?
- Where are most bugs located?
- Which features are error-prone?

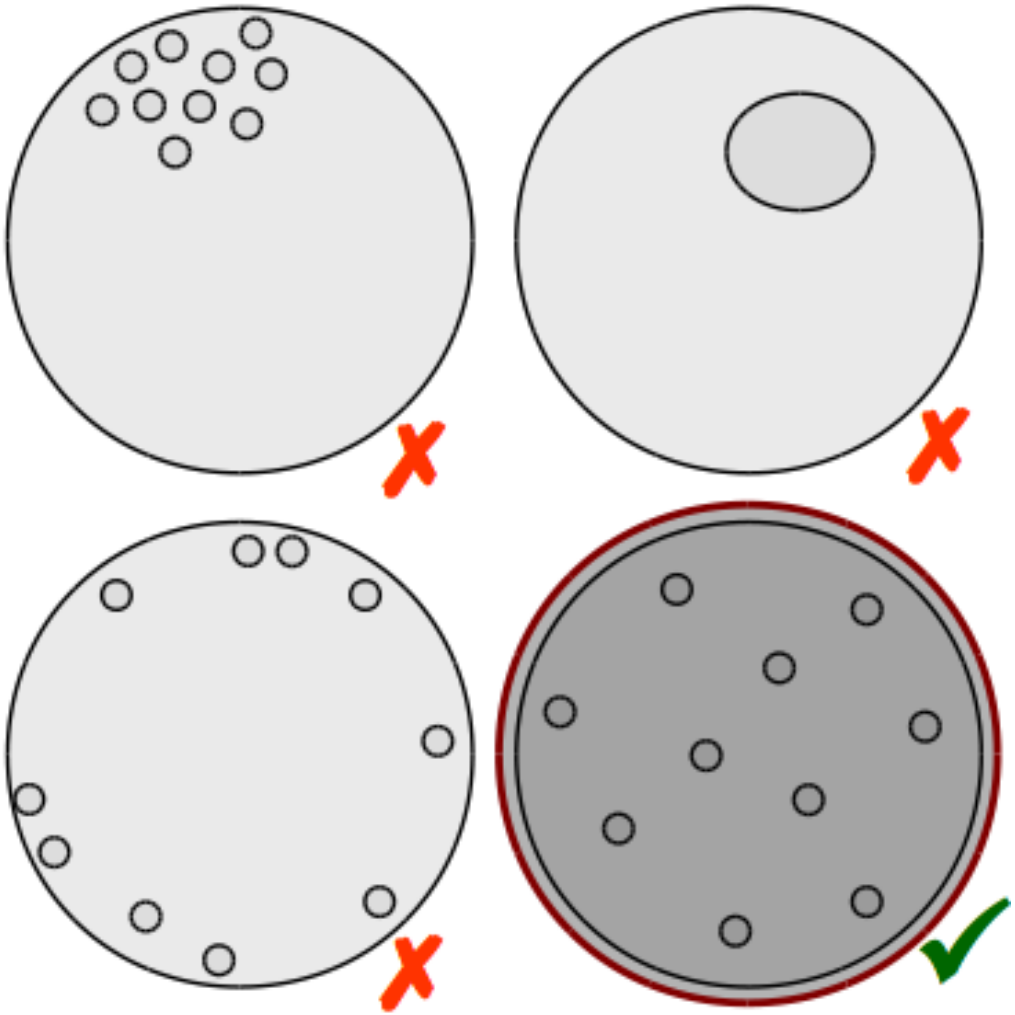
Metric

- **A representative sample of configurations**



Metric

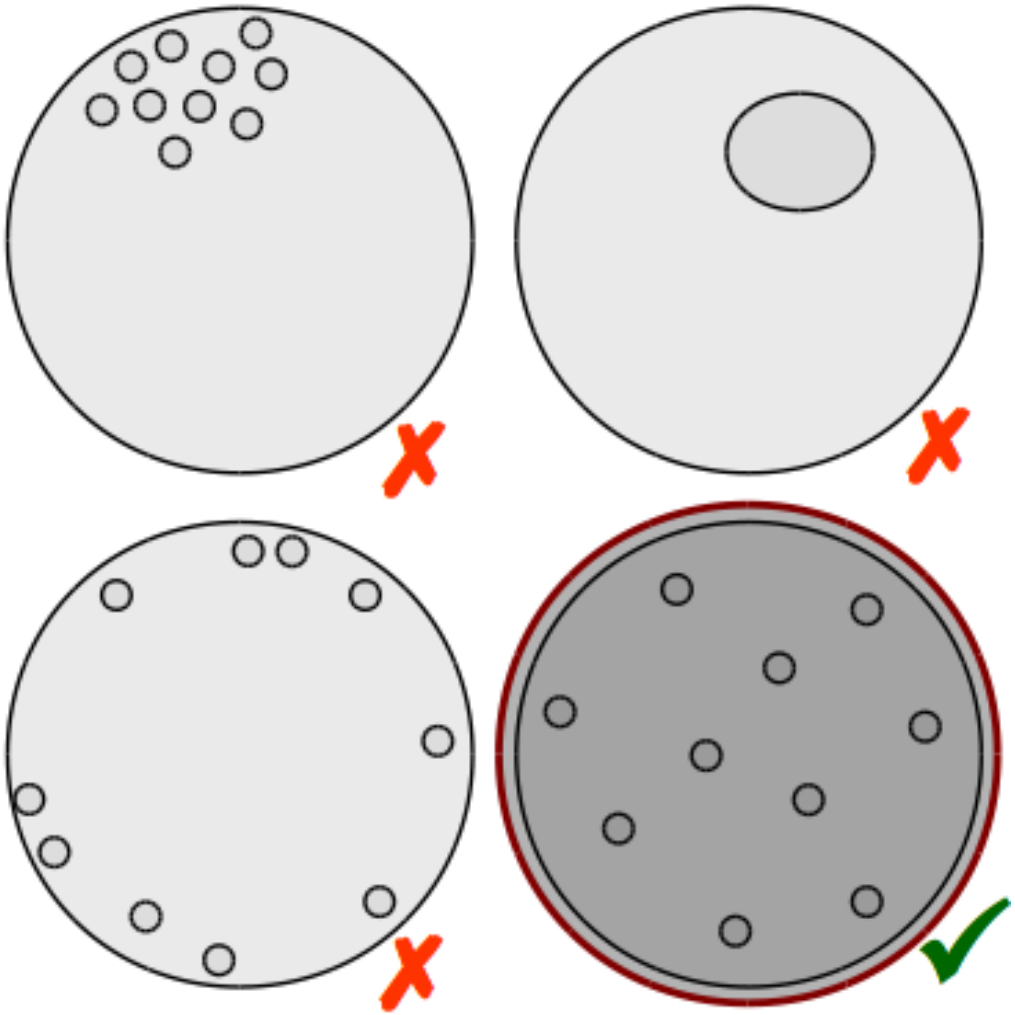
- **A representative sample of configurations**



- If it is representative, I can generalize

Metric

- **A representative sample of configurations**



- If it is representative, I can generalize
- Need a method of generating very random configurations from the feature model

Research Questions - Revisited

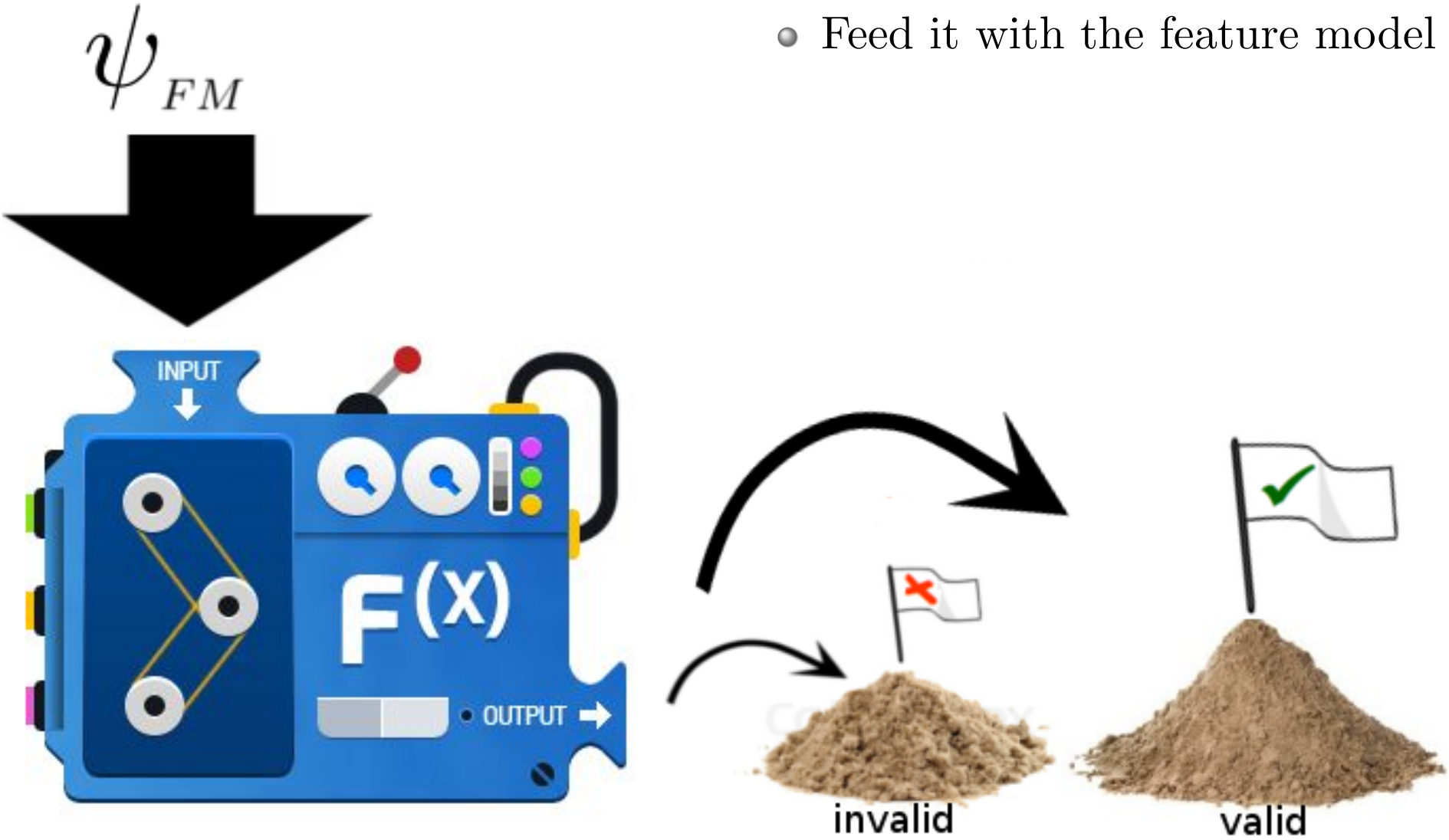
- How many bugs?
- What are the most common types of bugs?
- Where are most bugs?
- Which features are error-prone?

Research Questions - Revisited

- How many bugs?
- What are the most common types of bugs?
- Where are most bugs?
- Which features are error-prone?
- ... Additional questions?
- ... Prioritization?
- ... Take a few minutes to brainstorm about it
- (One-disabled vs. randconfig)

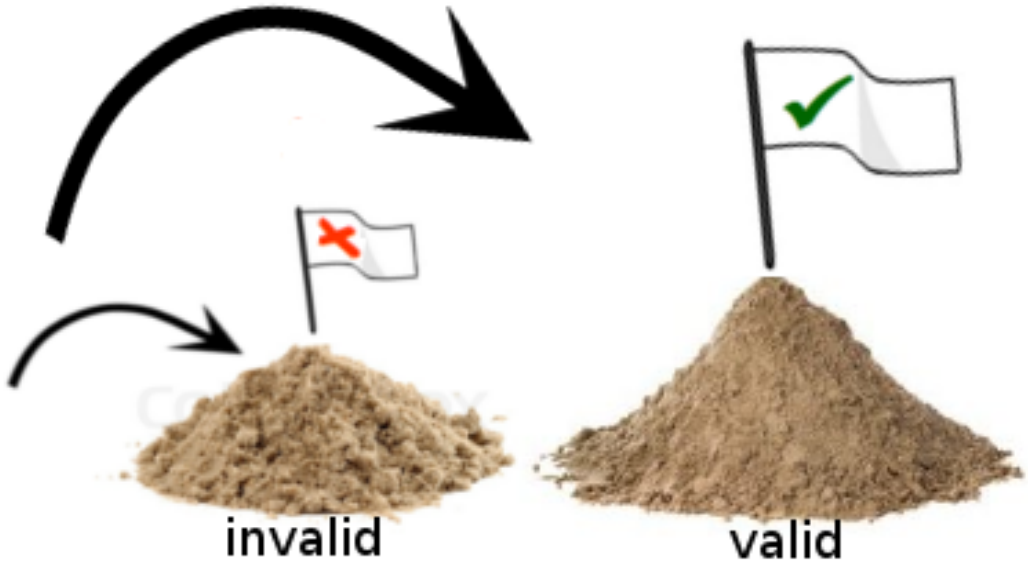
Methodology

- Feed it with the feature model

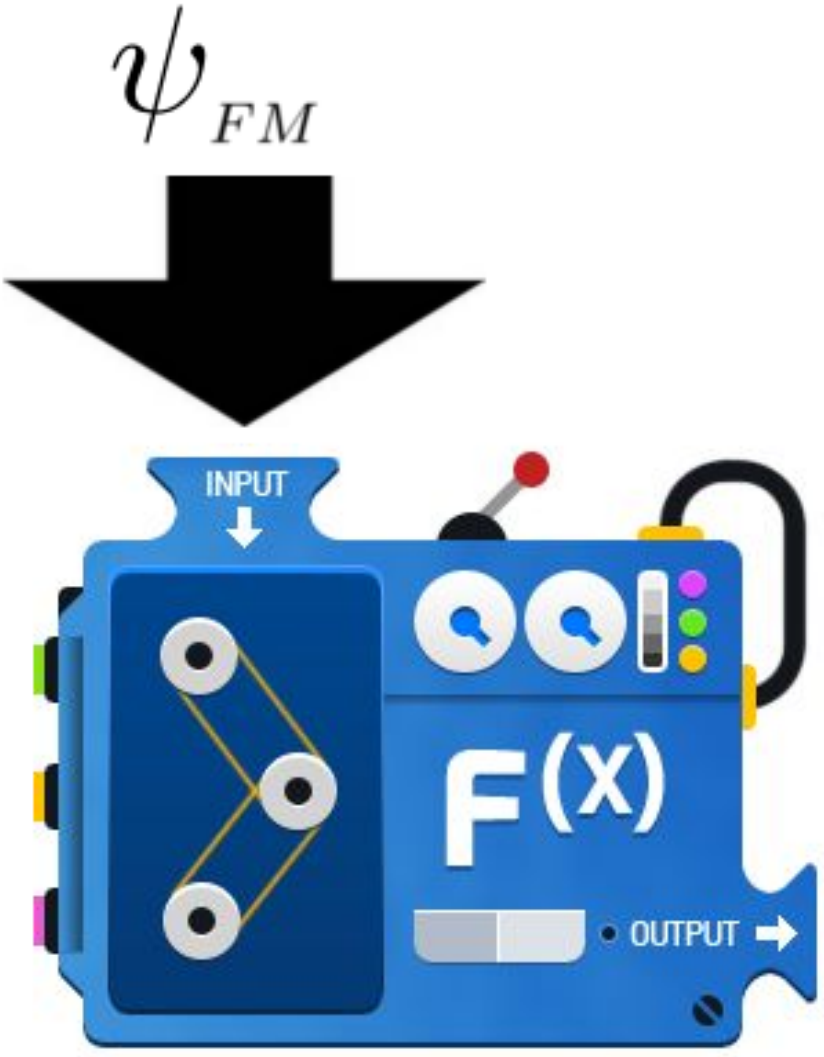


Methodology

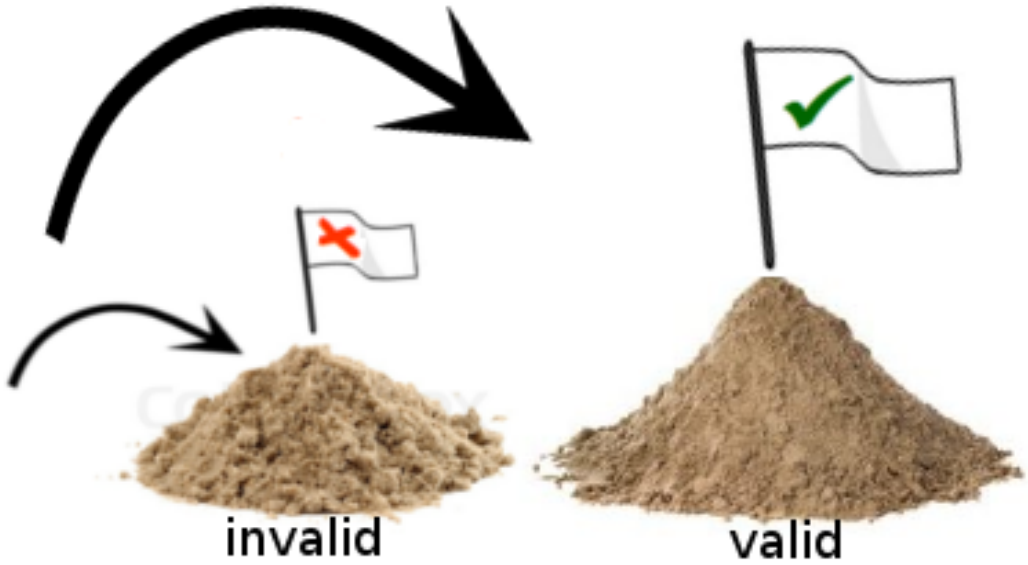
- Feed it with the feature model
- Collect κ_{random}



Methodology



- Feed it with the feature model
- Collect κ_{random}
- Assure $\kappa_{random} \in \mathbb{K}$



Methodology

Exploit existing randconfig

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

randconfig bias

```
config A
    bool

config B
    bool
    depends on A
```

randconfig bias

```
config A
    bool

config B
    bool
    depends on A
```

- Possible configurations with *randconfig*'s probabilities are:
  - () ***50%***
  - (A) ***25%***
  - (A,B) ***25%***

randconfig bias

```
config A
    bool

config B
    bool
    depends on A
```

- Possible configurations with *randconfig*'s probabilities are:
  - () ***50%***
  - (A) ***25%***
  - (A,B) ***25%***
- This is biased, and not representative.

randconfig bias

```
config A
    bool

config B
    bool
    depends on A
```

- We would prefer a totally equal probability:
  - () ***33%***
  - (A) ***33%***
  - (A,B) ***33%***

randconfig bias

```
config A
    bool

config B
    bool
    depends on A
```

- We would prefer a totally equal probability:
  - () ***33%***
  - (A) ***33%***
  - (A,B) ***33%***
- And so maybe we could invent something

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies
- Check for validity

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies
- Check for validity
  - Kconfig tool
  - SAT checker

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies
- Check for validity
  - Kconfig tool
  - SAT checker
- Uniform distribution

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies
- Check for validity
  - Kconfig tool
  - SAT checker
- Uniform distribution
- May take much time

Methodology

Exploit existing randconfig

- Concatenate Kconfig files
- Permute the order
- Flatten the structure
- Run *randconfig*
- Still biased in the coin flipping !
- See following Kconfig example

Invent elvisconfig

- Same idea as *randconfig*
- Try to fix bias in coin flipping
- Need to know #configs(A) vs. #configs(¬A)
- May be impossible

Generate 'n' Filter

- Flip a coin for every feature
- Do not visit dependencies
- Check for validity
  - Kconfig tool
  - SAT checker
- Uniform distribution
- May take much time

Other ideas? Best solution?

Analysing - Data collection

- Analyse one random configuration

Analysing - Data collection

- Analyse one random configuration (with **gcc**)

Analysing - Data collection

- Analyse one random configuration (with **gcc**)
- Save all kinds of information
  - Configuration.

Analysing - Data collection

- Analyse one random configuration (with **gcc**)
- Save all kinds of information
  - Configuration.
  - Errors / warnings

Analysing - Data collection

- Analyse one random configuration (with **gcc**)
- Save all kinds of information
  - Configuration.
  - Errors / warnings
  - Program versions (Linux / gcc)

Analysing - Data collection

- Analyse one random configuration (with **gcc**)
- Save all kinds of information
  - Configuration.
  - Errors / warnings
  - Program versions (Linux / gcc)
  - Make a bug reproducable

Analysing - Data collection

- Analyse one random configuration (with **gcc**)
- Save all kinds of information
  - Configuration.
  - Errors / warnings
  - Program versions (Linux / gcc)
  - Make a bug reproducable
- Do it over and over again (with a new random configuration)

Kconfig

- The language of the feature model in Linux (Busybox and others)

Kconfig

- The language of the feature model in Linux (Busybox and others)
  - Grammar
  - Data types
  - Transformation

Kconfig - Grammar

```
K ->      "config" ID TYPE OPT*
        | "if" ID K "endif"
        | "choice" K+ "endchoice"

TYPE ->   "bool" | "tristate" | "string" | "int"

OPT  ->   "default" STRING
        | "range" INT
        | "depends on" ID
        | "select" ID
```

Kconfig - Data types

```
config A
    bool
config B
    tristate
config C
    int
    range 5 15
config D
    hex
    default "c0000000"
config E
    string
    default "FOO"
```

y (ENABLED), n (DISABLED)

y, n, module

integers

hexadecimals

string

Kconfig - Transformation

```
config A
    bool

if A
    config B
        bool

    config C
        bool
endif
```

Transformation

```
config A
    bool

config B
    bool
    depends on A

config C
    bool
    depends on A
```

Threats to validity

- Sample not being representative
- Not getting enough bugs

Threats to validity

- Sample not being representative
- Not getting enough bugs
- Can you think of more?

Bye

Thank you for your time.