
[Alghero, Italy]

How Configurable is Linux?

On the Challenges of Analyzing the Kernel's Feature Model

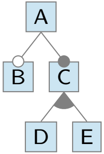
FOSD 2024 — April 9–12 — Eindhoven, Netherlands

Elias Kuiter[1], Chico Sundermann[2], Tobias Heß[2], Sebastian Krieter[3], Thomas Thüm[3], Gunter Saake[1]

University of Magdeburg[1], Ulm[2], Paderborn[3], Germany

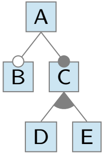
Configurability: A Fundamental Metric of Variability

A Small Feature Model . . .



Configurability: A Fundamental Metric of Variability

A Small Feature Model . . .

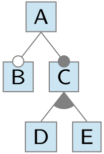


. . . and a Question to You

How **configurable** is this feature model?

Configurability: A Fundamental Metric of Variability

A Small Feature Model . . .



. . . and a Question to You

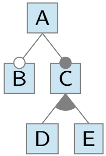
How **configurable** is this feature model?

Variability metrics as proxy questions:

- How many **features** does it have?
- How many valid **configurations** are there?
- count program variants (+ solution space)
- count distinct program variants (+ binary diff)
- count t-wise interactions, . . .

Configurability: A Fundamental Metric of Variability

A Small Feature Model ...



... and a Question to You

How **configurable** is this feature model?

Variability metrics as proxy questions:

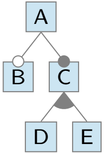
- How many **features** does it have?
- How many valid **configurations** are there?
- count program variants (+ solution space)
- count distinct program variants (+ binary diff)
- count t-wise interactions, ...

How Is This Relevant?

- judge **complexity** of feature models [Kuiter et al. 2024]
- **ground truth** for facilitated decision-making
- can also be applied to **subsystems** and **evolution**

Configurability: A Fundamental Metric of Variability

A Small Feature Model . . .



. . . and a Question to You

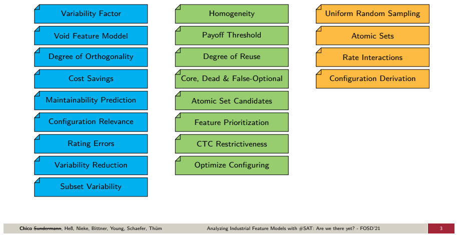
How **configurable** is this feature model?

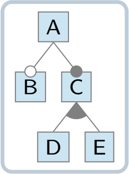
Variability metrics as proxy questions:

- How many **features** does it have?
- How many valid **configurations** are there?
- count program variants (+ solution space)
- count distinct program variants (+ binary diff)
- count *t*-wise interactions, . . .

How Is This Relevant?

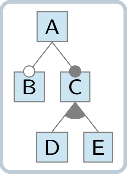
- judge **complexity** of feature models [Kuiter et al. 2024]
- **ground truth** for facilitated decision-making
- can also be applied to **subsystems** and **evolution**
- **#features** is fundamental, often stated in papers
- many applications for **#cfg**'s [Sundermann et al. 2021]



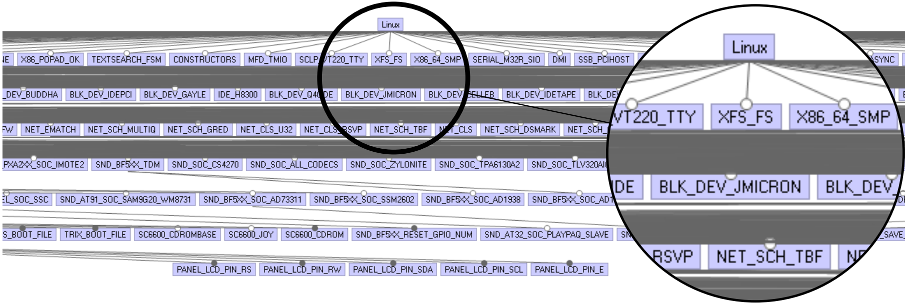


vs.

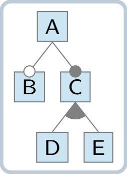
Linux: The End Boss of Feature-Model Analysis?



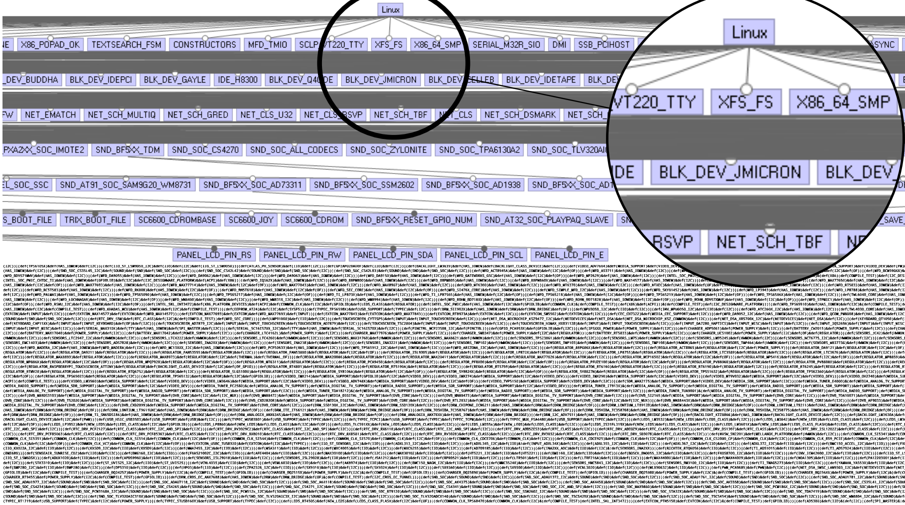
vs.



Linux: The End Boss of Feature-Model Analysis?



vs.



Literature on the Linux Kernel

Year	Revision	Architecture	Reported #Features	Reported #Configurations
2002	v2.5.45	*	—	—
2005	v2.6.12	i386	3,284	—
2008	v2.6.28	*	5,426	—
	v2.6.28	x86	5,321 5,323 6,888	—
	v2.6.28	x86?	5,701 6,888	—
	v2.6.28?	x86?	6,888	—
2009	v2.6.32	x86	6,319 6,320 60,072	—
2010	v2.6.33	x86	6,467 6,559 6,918 62,482	—
	v2.6.33?	x86?	5,913	—
2011	v3.1	*	11,691	—
2015	v4.0	x86	11,135	—
2016	v4.4	x86	15,500	—
2018	v4.18	x86	13,379 22,352	—
2020	v5.8	x86	14,817	—
2024	v6.7	*	—	—

(references omitted)

Literature on the Linux Kernel

Year	Revision	Architecture	Reported #Features	Reported #Configurations
2002	v2.5.45	*	—	—
2005	v2.6.12	i386	3,284	—
2008	v2.6.28	*	5,426	—
	v2.6.28	x86	5,321 5,323 6,888	—
	v2.6.28	x86?	5,701 6,888	—
	v2.6.28?	x86?	6,888	—
2009	v2.6.32	x86	6,319 6,320 60,072	—
2010	v2.6.33	x86	6,467 6,559 6,918 62,482	—
	v2.6.33?	x86?	5,913	—
2011	v3.1	*	11,691	—
2015	v4.0	x86	11,135	—
2016	v4.4	x86	15,500	—
2018	v4.18	x86	13,379 22,352	—
2020	v5.8	x86	14,817	—
2024	v6.7	*	—	—

- #features **unknown** for recent revisions
- #features **varies wildly** (impact on other analyses?)
- #configurations **unknown**

(references omitted)

Literature on the Linux Kernel

Year	Revision	Architecture	Reported #Features	Reported #Configurations
2002	v2.5.45	*	—	—
2005	v2.6.12	i386	3,284	—
2008	v2.6.28	*	5,426	—
	v2.6.28	x86	5,321 5,323 6,888	—
	v2.6.28	x86?	5,701 6,888	—
	v2.6.28?	x86?	6,888	—
2009	v2.6.32	x86	6,319 6,320 60,072	—
2010	v2.6.33	x86	6,467 6,559 6,918 62,482	—
	v2.6.33?	x86?	5,913	—
2011	v3.1	*	11,691	—
2015	v4.0	x86	11,135	—
2016	v4.4	x86	15,500	—
2018	v4.18	x86	13,379 22,352	—
2020	v5.8	x86	14,817	—
2024	v6.7	*	—	—

- #features **unknown** for recent revisions
- #features **varies wildly** (impact on other analyses?)
- #configurations **unknown**

Our Goals

- which **factors** influence this?
- which results are **accurate**?
- when is it **too hard**?

(references omitted)

Choosing and Analyzing the Feature Model

"are we talking about **the same** feature model?"

Choosing and Analyzing the Feature Model

"are we talking about **the same** feature model?"

1. Source Tree

⇒ here: **mainline kernel** (/torvalds/Linux)

Name

pub/scm/bluetooth
- bluetooth-next.git
- bluez-hcidump.git
- bluez.git
- obexd.git
- sbc.git

pub/scm/boot
- dracut/dracut.git
- efilinux/efilinux.git
- syslinux/syslinux.git

pub/scm/devel
- pahole/pahole.git
- sparse/chrisl/sparse.git
- sparse/sparse-dev.git
- sparse/sparse-logs.git
- sparse/sparse.git

pub/scm/docs
- docsko/ieee1394.git
- docsko/korg.git

Choosing and Analyzing the Feature Model

"are we talking about **the same** feature model?"

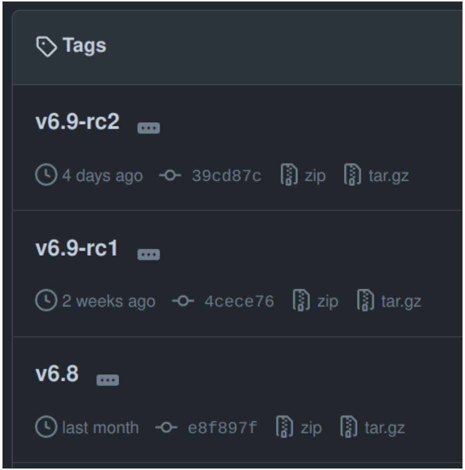
1. Source Tree

⇒ here: **mainline kernel** (/torvalds/Linux)

↓

2. Revision

⇒ here: **all releases** since 2002 (i.e., KConfig)



Choosing and Analyzing the Feature Model

"are we talking about **the same** feature model?"

1. Source Tree

⇒ here: **mainline kernel** (/torvalds/Linux)

↓

2. Revision

⇒ here: **all releases** since 2002 (i.e., KConfig)

↓

3. Architecture

⇒ here: **all architectures** (except for um)

Architecture	Subsumed Architecture
alpha	
arc	
arm, arm64	arm26
csky	
hexagon	
loongarch	
m68k (m68000)	m68knommu
microblaze	
mips	mips64
nios2	
openrisc	
parisc	parisc64
powerpc	ppc, ppc64
riscv	
s390 (z Systems)	s390x
sh (SuperH)	sh64
sparc	sparc32, sparc64
um (User Mode Linux)	
x86	i386, x86_64
xtensa	

Choosing and **Analyzing the Feature Model**

"are we analyzing the model **the same way**?"

Choosing and Analyzing the Feature Model

```
config X86_32
  def_bool y
  depends on !64BIT
  select ARCH_WANT_IPC_PARSE_VERSION

config SMP
  bool "Symmetric multi-processing support"
  help
    This enables support for systems with
    more than one CPU.

if X86_32
config X86_BIGSMP
  bool "Support for big SMP systems"
  depends on SMP
  help
    This option is needed for the systems
    that have more than 8 CPUs.
endif # X86_32
```

"are we analyzing the model **the same way**?"

4. Extraction

⇒ here: **KConfigReader** and **KClause (KMax)**

$$F = \{\mathtt{X86\_32}, \mathtt{64BIT}, \mathtt{SMP}, \mathtt{X86\_BIGSMP}\}$$
$$\phi = (\mathtt{X86\_32} \rightarrow \neg \mathtt{64BIT}) \wedge (\mathtt{X86\_BIGSMP} \rightarrow (\mathtt{X86\_32} \wedge \mathtt{SMP}))$$

Choosing and **Analyzing the Feature Model**

CNF Transformation [Kuiter et al. 2022]

$$\phi_{CNF} := tseitinCNF(\phi)$$

Backbone Transformation [Biere et al. 2023]

$$V_{dead} := \{v \in V \mid \neg SAT(\phi_{CNF} \land v)\}$$
$$V_{core} := \{v \in V \mid \neg SAT(\phi_{CNF} \land \neg v)\}$$
$$\phi_{CNF}^{back} := \phi_{CNF} \land \bigwedge_{v \in V_{dead}} \neg v \land \bigwedge_{v \in V_{core}} v$$

"are we analyzing the model **the same way**?"

4. Extraction

$\Rightarrow$ here: **KConfigReader** and **KClause (KMax)**

$\downarrow$

5. Transformation

$\Rightarrow$ here: **CNF** and **backbone transformation**

Choosing and Analyzing the Feature Model

#Features

$|F|$ (sort of)

#Configurations

$\#SAT(V, \phi)$ (sort of)

(more on this later)

"are we analyzing the model **the same way**?"

4. Extraction

$\Rightarrow$ here: **KConfigReader** and **KClause (KMax)**

$\downarrow$

5. Transformation

$\Rightarrow$ here: **CNF** and **backbone transformation**

$\downarrow$

6. Analysis

$\Rightarrow$ here: **#features** and **#configurations**

Evaluation

Choosing and Analyzing the Feature Model

1. Source Tree

only mainline kernel

2. Revision

all releases $\geq$ 2002

3. Architecture

all but `um`

4. Extraction

KConfigReader, KClause

5. Transformation

CNF, backbone

6. Analysis

#features, #configurations

Research Questions

Evaluation

Choosing and Analyzing the Feature Model

1. Source Tree
only mainline kernel

2. Revision
all releases $\geq$ 2002

3. Architecture
all but `um`

4. Extraction
KConfigReader, KClause

5. Transformation
CNF, backbone

6. Analysis
#features, #configurations

Research Questions

RQ$_1$ **When** can we count features/configurations?

Evaluation

Choosing and Analyzing the Feature Model

1. Source Tree

only mainline kernel

2. Revision

all releases $\geq$ 2002

3. Architecture

all but `um`

4. Extraction

KConfigReader, KClause

5. Transformation

CNF, backbone

6. Analysis

#features, #configurations

Research Questions

RQ$_1$ **When** can we count features/configurations?

RQ$_2$ How to count **features**?
Influence of **revision**, **architecture**, **extractor**?

Evaluation

Choosing and Analyzing the Feature Model

1. Source Tree
only mainline kernel

2. Revision
all releases $\geq$ 2002

3. Architecture
all but `um`

4. Extraction
KConfigReader, KClause

5. Transformation
CNF, backbone

6. Analysis
#features, #configurations

Research Questions

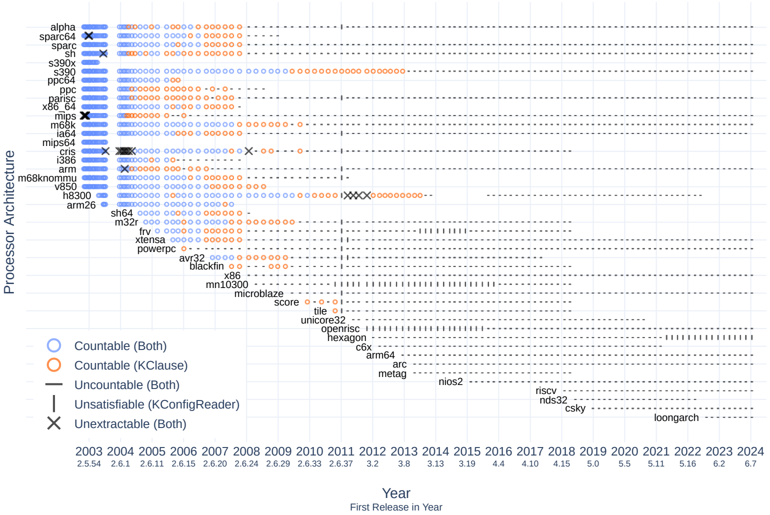
RQ_1 **When** can we count features/configurations?

RQ_2 How to count **features**?
Influence of **revision**, **architecture**, **extractor**?

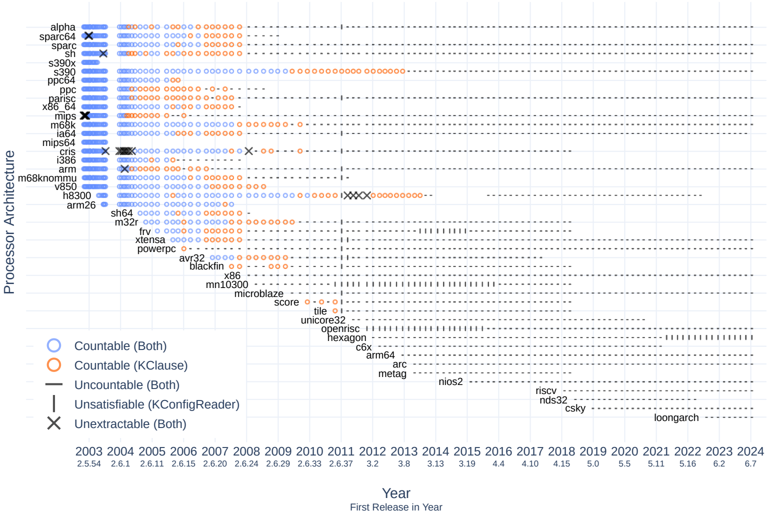
RQ_3 How to count **configurations**?
Influence of **revision**, **architecture**, **extractor**?

RQ_1: When Can We Count Features and Configurations?

RQ$_1$: When Can We Count Features and Configurations?



RQ$_1$: When Can We Count Features and Configurations?



100% #features

35.8% #configurations

$RQ_{2\text{–}3}$: How to Count Features and Configurations?

$\text{RQ}_{2\text{–}3}$: How to Count Features and Configurations?

"Isn't this just the cardinality of F?" Yes-ish. (60%)

$RQ_{2\text{–}3}$: How to Count Features and Configurations?

"Isn't this just the cardinality of F?" Yes-ish. (60%)

Improved #Features

- begin with the formula's **variables**
- remove **auxiliary variables** (from *tseitinCNF*)
- remove **non-related variables** (i.e., modules and visibility conditions)
- remove **dead features**, which cannot be selected
- add **unconstrained features**, which can be selected freely
- cross-reference with features defined in **KConfig files**, remove non-Boolean variables

RQ_{2-3}: How to Count Features and Configurations?

"Isn't this just the cardinality of F?" Yes-ish. (60%)

"Isn't this just the answer to the #SAT query $\#SAT(V, \phi)$?" Yes-ish. $(1 : 10^{10})$

Improved #Features

- begin with the formula's **variables**
- remove **auxiliary variables** (from *tseitinCNF*)
- remove **non-related variables** (i.e., modules and visibility conditions)
- remove **dead features**, which cannot be selected
- add **unconstrained features**, which can be selected freely
- cross-reference with features defined in **KConfig files**, remove non-Boolean variables

RQ$_{2\text{–}3}$: How to Count Features and Configurations?

"Isn't this just the cardinality of F?" Yes-ish. (60%)

"Isn't this just the answer to the #SAT query $\#SAT(V, \phi)$?" Yes-ish. ($1:10^{10}$)

Improved #Features

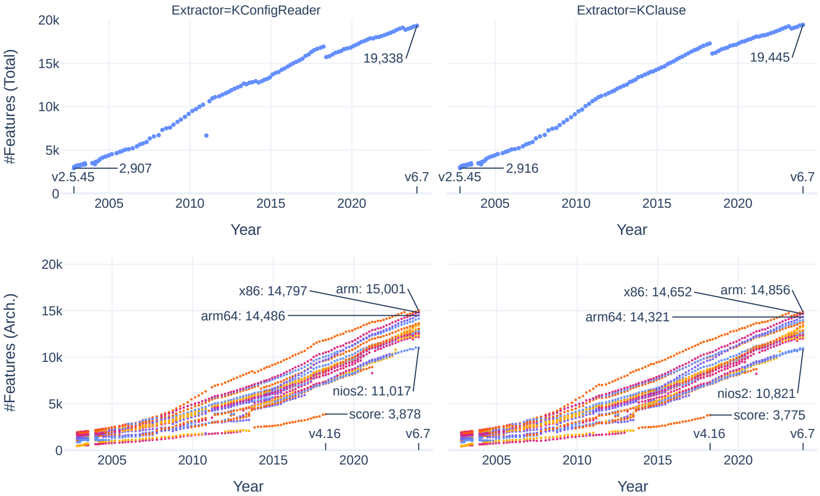
- begin with the formula's **variables**
- remove **auxiliary variables** (from *tseitinCNF*)
- remove **non-related variables** (i.e., modules and visibility conditions)
- remove **dead features**, which cannot be selected
- add **unconstrained features**, which can be selected freely
- cross-reference with features defined in **KConfig files**, remove non-Boolean variables

Improved #Configurations

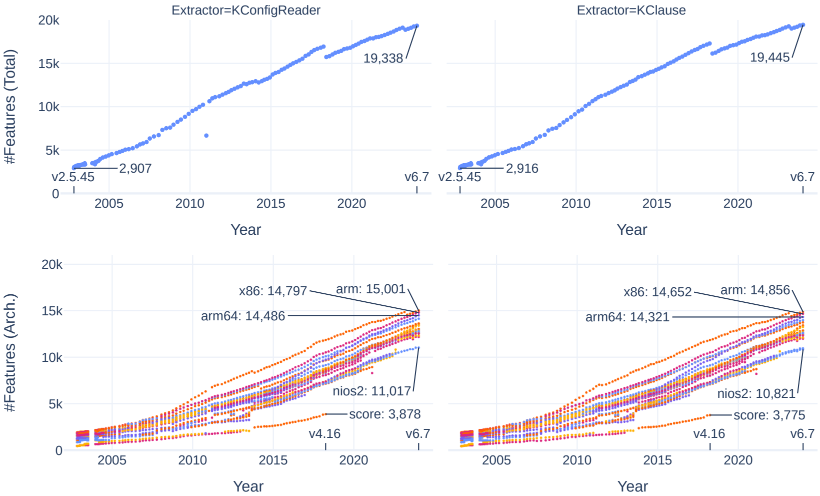
- begin with the formula's **model count**
- add **unconstrained features**, which can be selected freely ($*2^{|F_{unconstrained}|}$)

RQ_2: How to Count Features? – Results

RQ$_2$: How to Count Features? – Results



RQ$_2$: How to Count Features? – Results



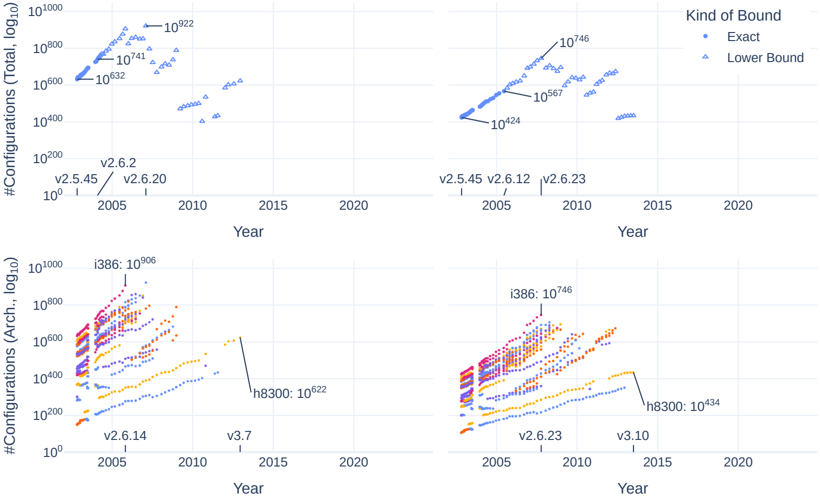
sustained **linear growth** (still!)

a given **architecture** only contains half of the features

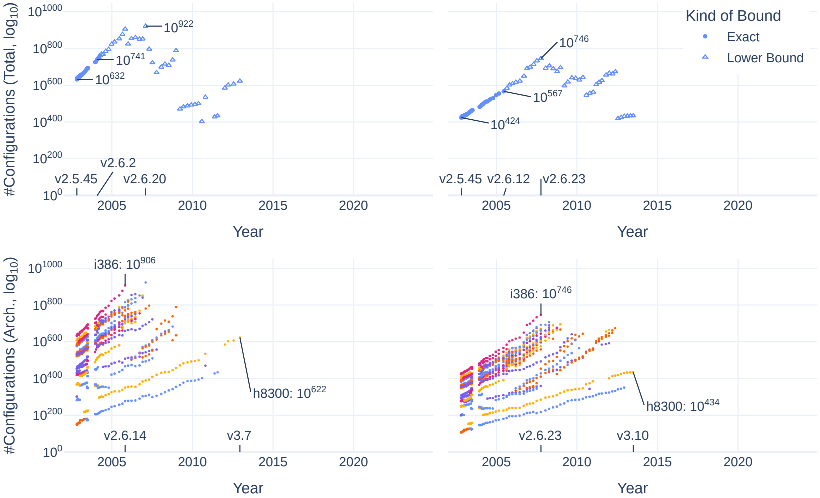
does not depend on **extractor**

RQ_3: How to Count Configurations? – Results

RQ$_3$: How to Count Configurations? – Results



RQ$_3$: How to Count Configurations? – Results



exponential growth

only $^1/_{10^{130}}$ configurations are **valid**

depends on **extractor** ($log_{10} * 1.5$)

pushes the limits of state-of-the-art #SAT solvers

Goodies for You

Goodies for You

torte: Feature-Model Experiments à La Carte 🍰

declarative, fully automated, reproducible

1. Source Tree

any

2. Revision

any $\geq$ 2002

3. Architecture

any but `um`

4. Extraction

KConfigReader,
KClause

5. Transformation

FeatureIDE, FeatJAR,
z3, clausy, CaDiBack

6. Analysis

the above + SATGraf,
dozens of solvers

Goodies for You

torte: Feature-Model Experiments à La Carte

declarative, fully automated, reproducible

1. Source Tree

any

2. Revision

any $\geq$ 2002

3. Architecture

any but `um`

4. Extraction

KConfigReader, KClause

5. Transformation

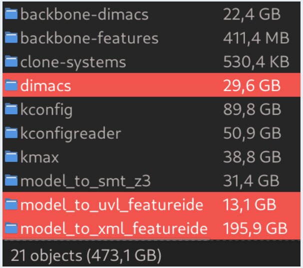
FeatureIDE, FeatJAR, z3, clausy, CaDiBack

6. Analysis

the above + SATGraf, dozens of solvers

→

Comprehensive Dataset



$>$ **3000 feature models** of the Linux kernel

(weekly sample also available)

(for now, available on request)

Goodies for You

torte: Feature-Model Experiments à La Carte

declarative, fully automated, reproducible

1. Source Tree

any

2. Revision

any $\geq$ 2002

3. Architecture

any but `um`

4. Extraction

KConfigReader, KClause

5. Transformation

FeatureIDE, FeatJAR, z3, clausy, CaDiBack

6. Analysis

the above + SATGraf, dozens of solvers

$\rightarrow$

Comprehensive Dataset

backbone-dimacs	22,4 GB
backbone-features	411,4 MB
clone-systems	530,4 KB
dimacs	29,6 GB
kconfig	89,8 GB
kconfigreader	50,9 GB
kmax	38,8 GB
model_to_smt_z3	31,4 GB
model_to_uvl_featureide	13,1 GB
model_to_xml_featureide	195,9 GB
21 objects (473,1 GB)	

$>$ **3000 feature models** of the Linux kernel

(weekly sample also available)

(for now, available on request)

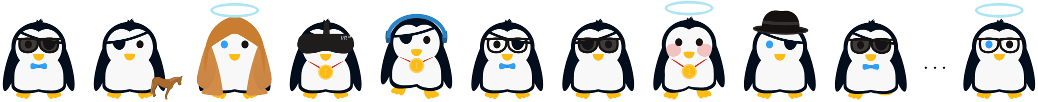
`curl -s https://ekuiter.github.io/torte/ | sh -s - linux-history-releases` (takes a few weeks!)

Conclusion

[LKML 2012]

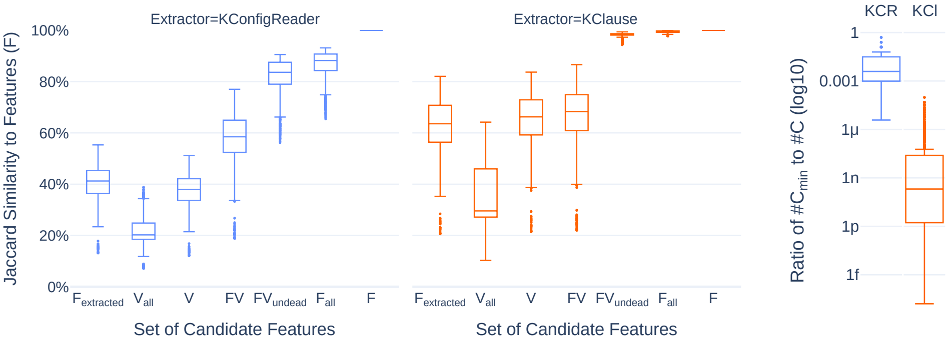
Date Sun, 1 Apr 2012 00:33:21 +0800
From Paul E. McKenney
Subject [PATCH RFC] Simplify the Linux kernel by reducing its state space

Although there have been numerous complaints about the complexity of parallel programming, the plain truth is that the incremental complexity of parallel programming over that of sequential programming is not as large as is commonly believed. Despite that you might have heard, the mind-numbing complexity of modern computer systems is not due so much to there being multiple CPUs, but rather to there being any CPUs at all. In short, for the ultimate in computer-system simplicity, the optimal choice is NR_CPUS=0.
This commit therefore limits kernel builds to zero CPUs. This change has the beneficial side effect of rendering all kernel bugs harmless. Furthermore, this commit enables additional beneficial changes, for example, the removal of those parts of the kernel that are not needed when there are zero CPUs.

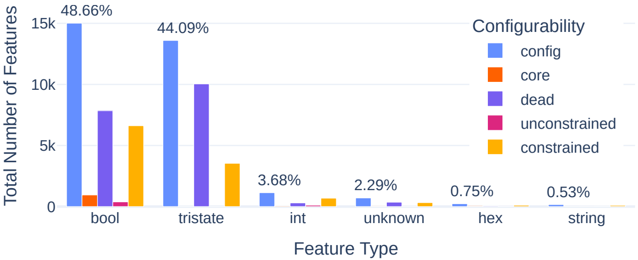


Thank you for listening!

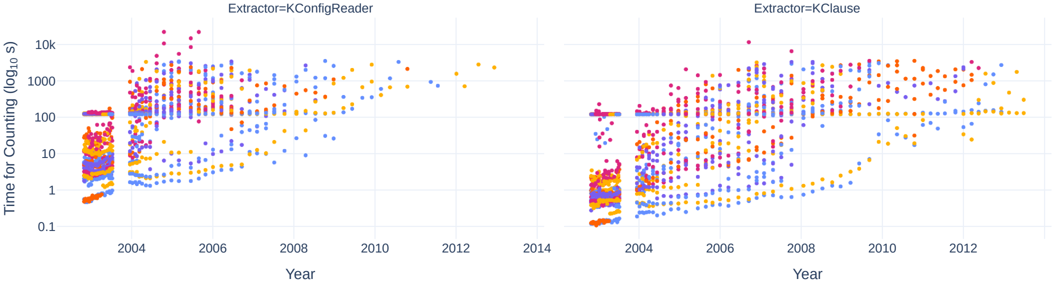
Feature and Configuration Candidates



Feature Types



Model Count Time



Failed Attempts and Future Directions

- approximate model counting: works worse then standard #SAT
- model approximate counting: use exact counter on approximate model (where hard constraints are omitted)
- knowledge compilation (BDD, d-DNNF, . . .): still too hard [Thüm 2020, Sundermann et al. 2023]
- incremental counting (count #SAT differences): grow exponentially as well
- prime factorization (shorten the #SAT ratio): is this possible?
- non-Boolean variability: what encoding is actually needed in which use case? (Boolean, bit-blasting, equivalence classes, solution-space model, . . .)
- architecture unification (eliminate architecture as a threat of validity)
- future projection (make predictions about the development of Linux)