ESEC/FSE 2021

Aug. 25th-26th, 2021

Finding Broken Linux Configuration Specifications by Statically Analyzing the Kconfig Language

Jeho Oh (UT Austin)

Necip Fazıl Yıldıran (UCF)

Julian Braha (UCF)

Paul Gazzillo (UCF)

UCF
UNIVERSITY OF CENTRAL FLORIDA

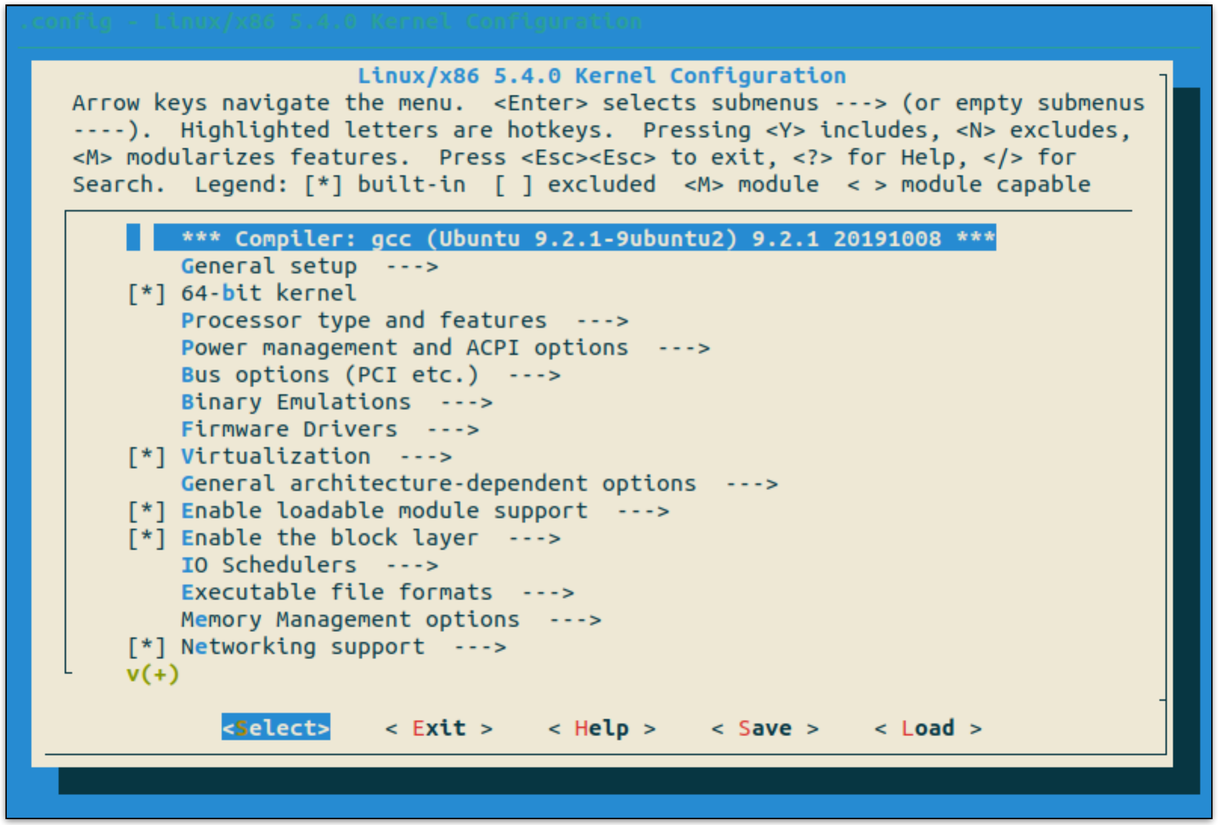
highly-configurable software underpins much of our computing infrastructure

linux kernel

- 15,000+ configuration options

billions of devices

developers provide configuration specifications for intended combinations of configuration options



large configuration specifications make maintenance harder

- meant to ensure correct software configuration
  - about 140,000 lines of kconfig
- complex semantics beyond feature modeling, e.g.,
  - typed options
  - invisible options
  - automated selection of options
  - user-interface constructs

kconfig language example

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB
```

kconfig language example

declaring the option

```
config TOUCHSCREEN_ADC
	tristate
	prompt "Generic ADC based touchscreen"
	depends on IIO && INPUT_TOUCHSCREEN
	select IIO_BUFFER_CB
```

kconfig language example

```
config TOUCHSCREEN_ADC
	tristate
	prompt "Generic ADC based touchscreen"
	depends on IIO && INPUT_TOUCHSCREEN
	select IIO_BUFFER_CB
```

giving it a type

kconfig language example

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB
```

text description for the user interface

kconfig language example

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB
```

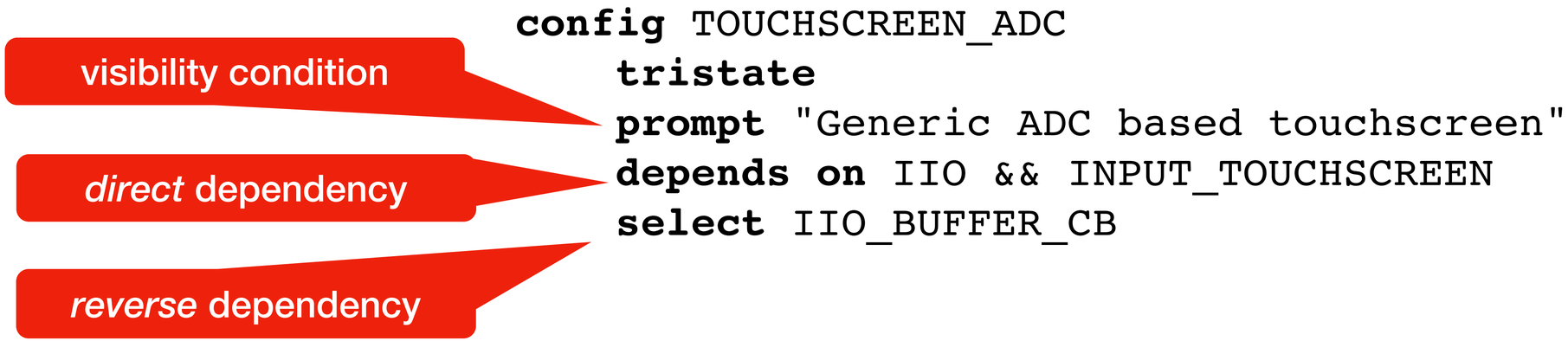
requires IIO and INPUT_TOUCHSCREEN

kconfig language example

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB
```

automatically turns on IIO_BUFFER_CB

kconfig language example



the unmet dependency bug

> “**select should be used with care. select will force a symbol to a value without visiting the dependencies.** By abusing select you are able to select a symbol FOO even if FOO depends on BAR that is not set. In general use select only for non-visible symbols (no prompts anywhere) and for symbols with no dependencies. That will limit the usefulness but on the other hand avoid the illegal configurations all over”

https://www.kernel.org/doc/html/latest/kbuild/kconfig-language.html

an unmet dependency bug in the wild

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB

config IIO_BUFFER
    bool
    prompt "Enable buffer support within IIO"
    depends on IIO

config IIO_BUFFER_CB
    tristate
    prompt "IIO callback buffer"
    depends on IIO && IIO_BUFFER
```

an unmet dependency bug in the wild

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB

config IIO_BUFFER
    bool
    prompt "Enable buffer support within IIO"
    depends on IIO

config IIO_BUFFER_CB
    tristate
    prompt "IIO callback buffer"
    depends on IIO && IIO_BUFFER
```

IIO_BUFFER_CB
depends on
IIO and IIO_BUFFER

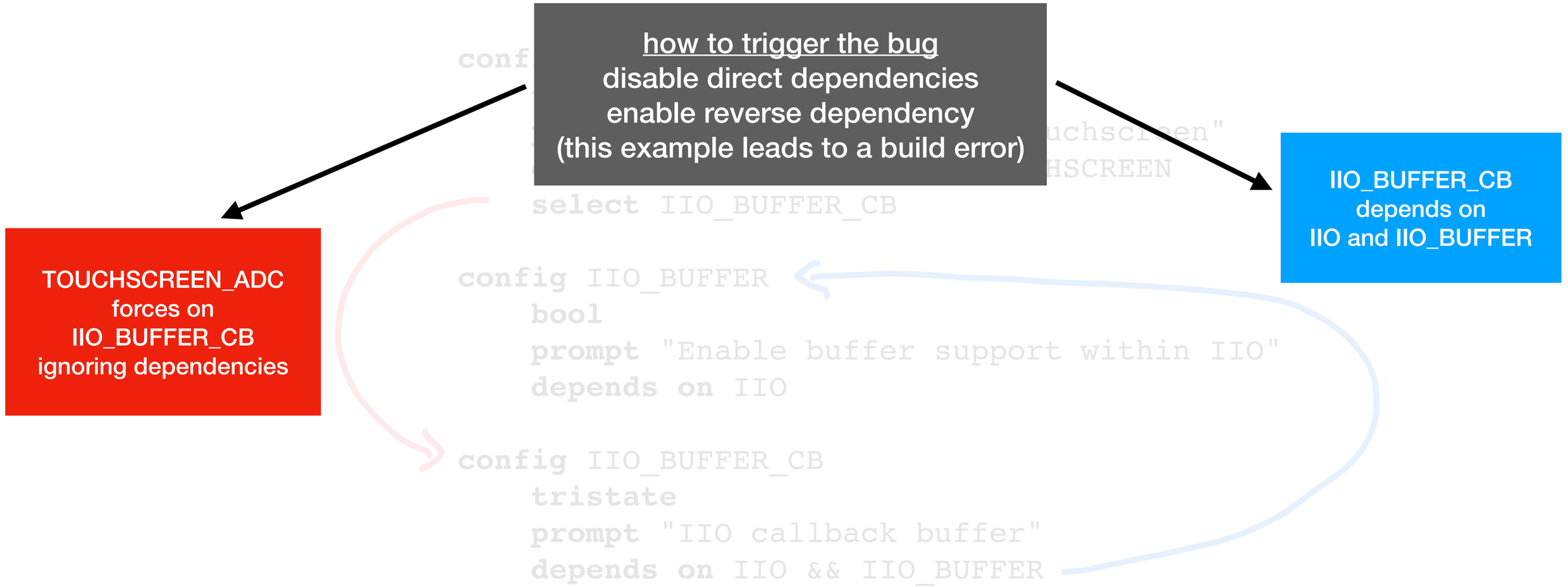
an unmet dependency bug in the wild

```
config TOUCHSCREEN_ADC
    tristate
    prompt "Generic ADC based touchscreen"
    depends on IIO && INPUT_TOUCHSCREEN
    select IIO_BUFFER_CB

config IIO_BUFFER
    bool
    prompt "Enable buffer support within IIO"
    depends on IIO

config IIO_BUFFER_CB
    tristate
    prompt "IIO callback buffer"
    depends on IIO && IIO_BUFFER
```

TOUCHSCREEN_ADC
forces on
IIO_BUFFER_CB
ignoring dependencies

IIO_BUFFER_CB
depends on
IIO and IIO_BUFFER

an unmet dependency bug in the wild



goal: automated analysis of kconfig

- large specification make scalability a challenge (~140,000 lines)
  - many uses of select constructs (~12,000)
- lots of configuration options (~15,000)
  - large space of possible input configurations to test
- buggy vs. safe configuration declarations look similar
  - complex networks of dependencies obscures behavior

our solution

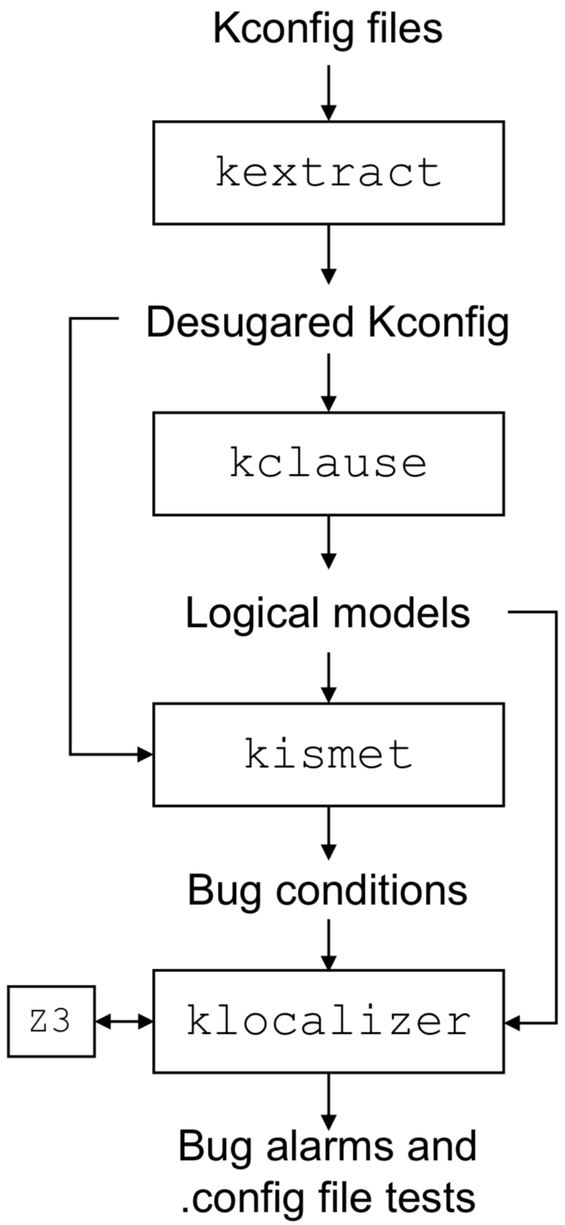
static analysis for kconfig using our new software model checking infrastructure

challenges to static analysis of kconfig

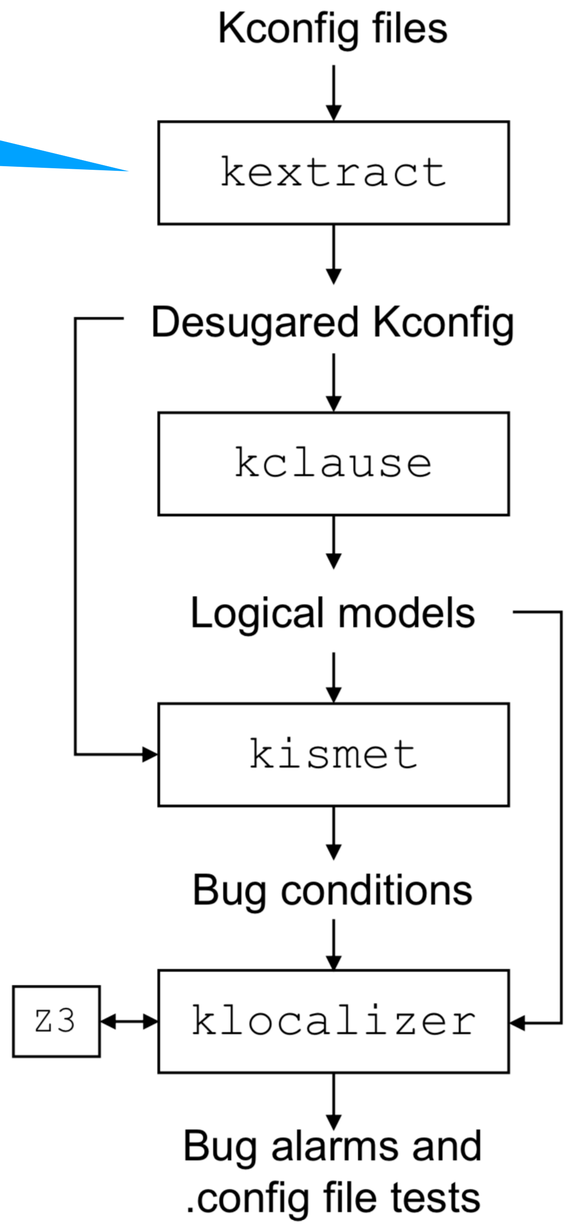
- no prior static analysis infrastructure
- insufficient existing work describing kconfig semantics
- scaling to large kconfig specifications (~140.000 lines for linux)
- kconfig syntax changes gradually over time as it is modified

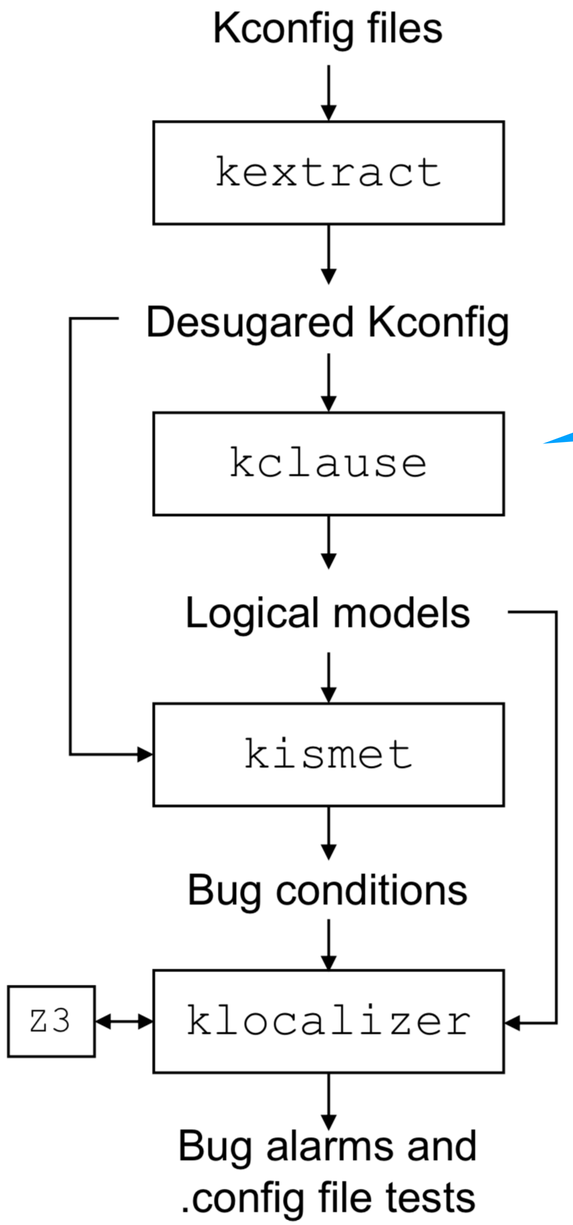
contributions

1. a formal semantics of kconfig
2. an verification-based unmet dependency finder with optimizations
3. an implementation of the semantics (kclause) and bug finder (kismet)
4. an evaluation of performance, precision, and impact

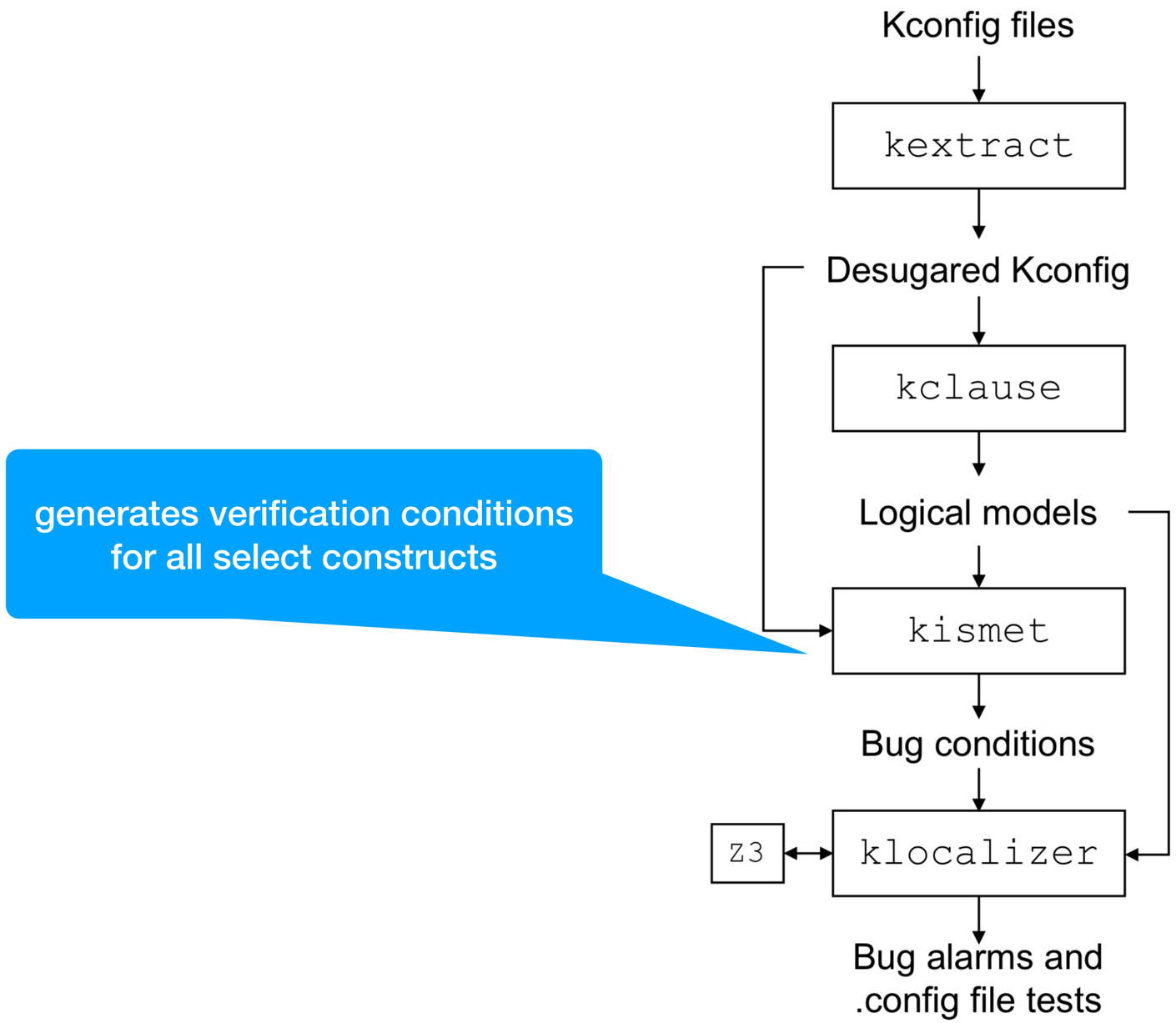

Kconfig files
kextract
Desugared Kconfig
kclause
Logical models
kismet
Bug conditions
Z3
klocalizer
Bug alarms and
.config file tests

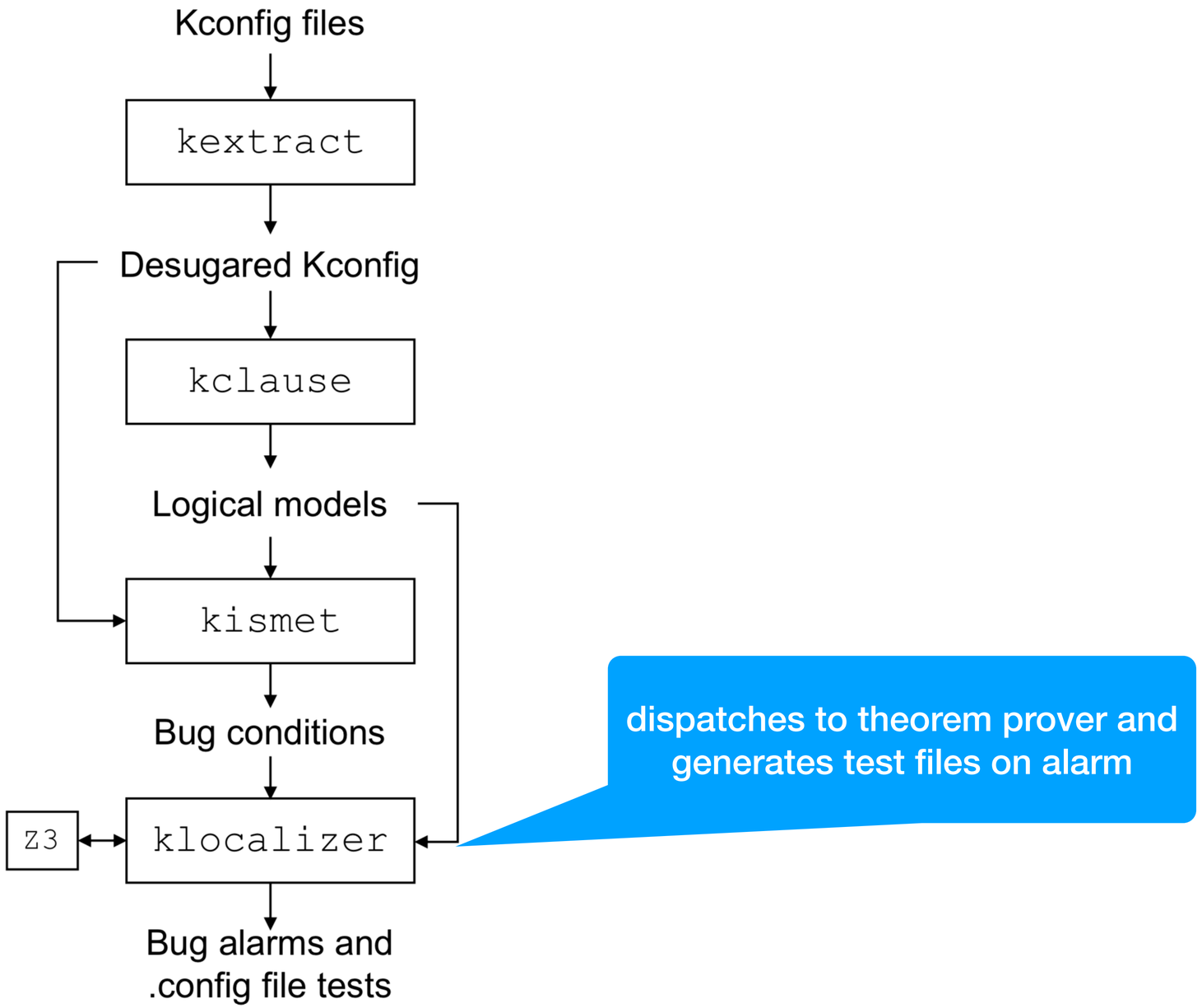
simplifies syntax, uses kconfig parser from linux source code

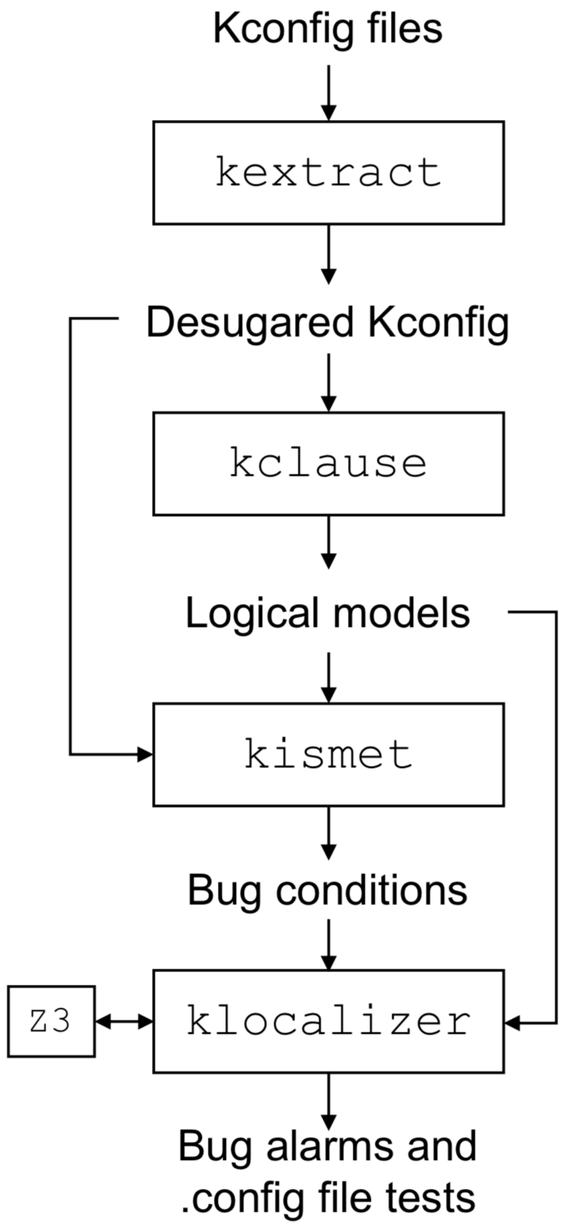




implements our semantics as translator to logic


Kconfig files
kextract
Desugared Kconfig
kclause
Logical models
kismet
Bug conditions
Z3
klocalizer
Bug alarms and
.config file tests
generates verification conditions
for all select constructs


Kconfig files
kextract
Desugared Kconfig
kclause
Logical models
kismet
Bug conditions
Z3
klocalizer
Bug alarms and
.config file tests
dispatches to theorem prover and
generates test files on alarm


Kconfig files
kextract
Desugared Kconfig
kclause
Logical models
kismet
Bug conditions
Z3
klocalizer
Bug alarms and
.config file tests

experimental setup

- search for unmet dependency bugs
- linux v5.4.4 source code
- 28 architecture families
- each has its own kconfig specification
  - though most contents are shared between architectures (hardware abstraction layer)
- run kismet on each of the 28
  - deduplicate results from shared select constructs

precision (true positives)

- 151 true positive bugs
  - 781 before deduplicating the 28 kconfig specifications
- bugs validated by automatically generating test configuration files
  - we convert solutions to bug verification condition to linux config file format
- 100% precision
  - for boolean and tristate options
  - we underapproximate non-boolean options

recall (false negatives)

- kismet deliberaly underapproximate for non-boolean options
- unknown ground truth (real-world linux specs)
- we generated and tested about 11,000,000 configuration files
  - used de-facto standard tool, randconfig, over several sequential months of time
- random testing found 8 true positives not found by kismet
  - kismet found 614 not found by randconfig

performance

- 37 to 90 minutes to run a kconfig specification
  - 10,014 to 12,744 select constructs analyzed for each specification
  - all 28 specifications: a little less than one sequential day
  - fast enough to run daily (speed of linux-next repo)
- more bugs found in one hour compared to random testing
  - recall/precision tradeoff for non-boolean underapproximation
  - useful complement to randconfig

impact

- we submitted bug reports (38) and patches for some bugs so far
  - limited by manual effort to patch, report, and converse with maintainers
- all reports accepted as true
  - at least one known and left in intentionally
  - some report still pending reply or resolution
- 15 patches mainlined so far

conclusion

- highly-configurable software is widespread
- problem: configuration specifications are large and complex
- our goal: automated static analyses for the kconfig specification language
- contributions:
  - kconfig formal semantics; an unmet dependency bug-finder; an implementation; an evaluation
- our tooling is fast and precise, has led to accepted patches in linux's kconfig specs

supported by NSF CCF-1941816 and CCF-1840934

try it out!
https://github.com/paulgazz/kmax
pip3 install kmax