Efficient, Yet Robust Extraction of Variability Information from Linux Makefiles

Andreas Ruprecht
andreas.ruprecht@fau.de

Valentin Rothberg
valentin.rothberg@lip6.fr

Daniel Lohmann
dl@cs.fau.de

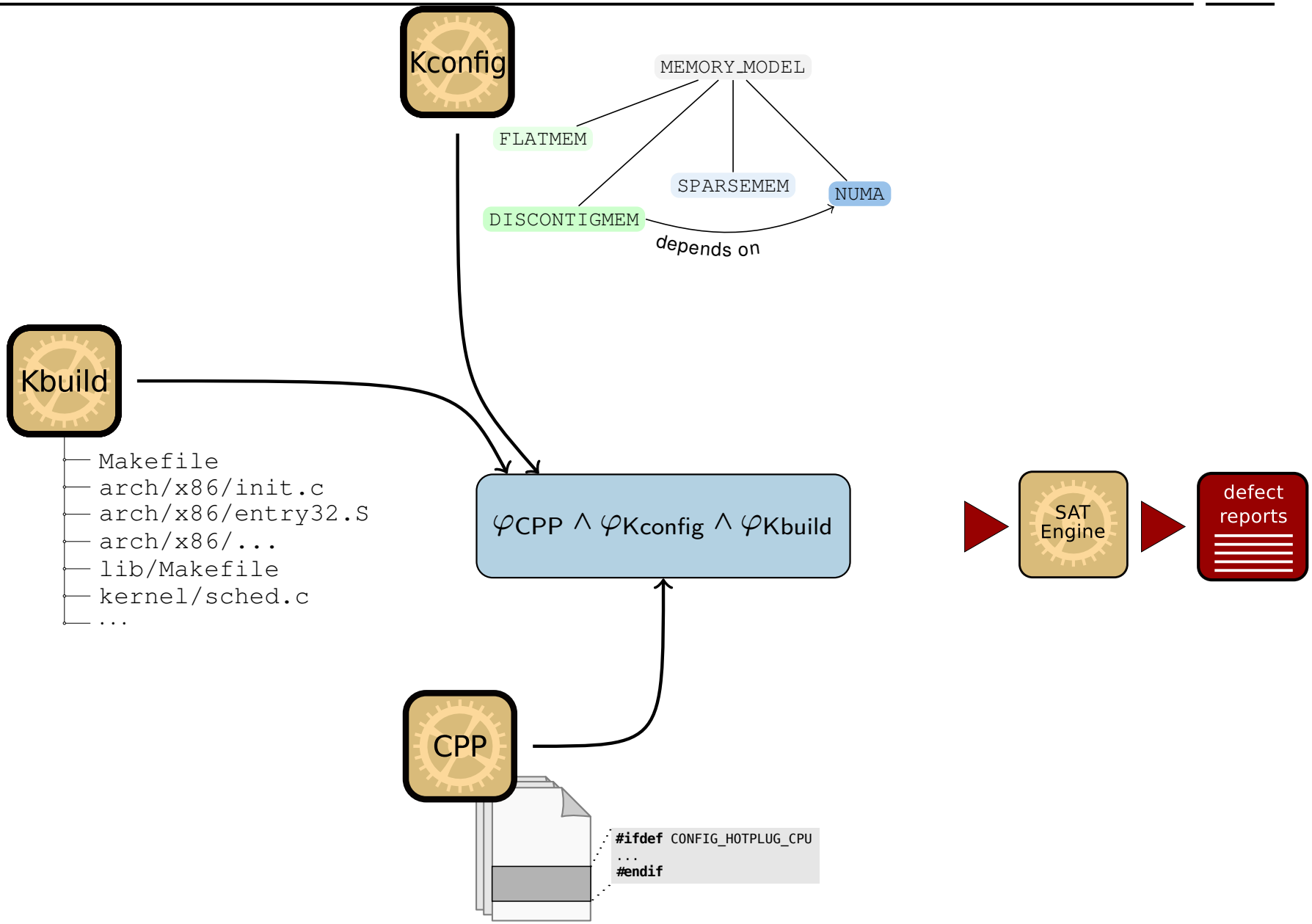
System Software Group
Friedrich-Alexander University Erlangen-Nürnberg (FAU)
and
Inria / LIP6 Paris

`https://cados.cs.fau.de`

FOSD Meeting '15

supported by DFG

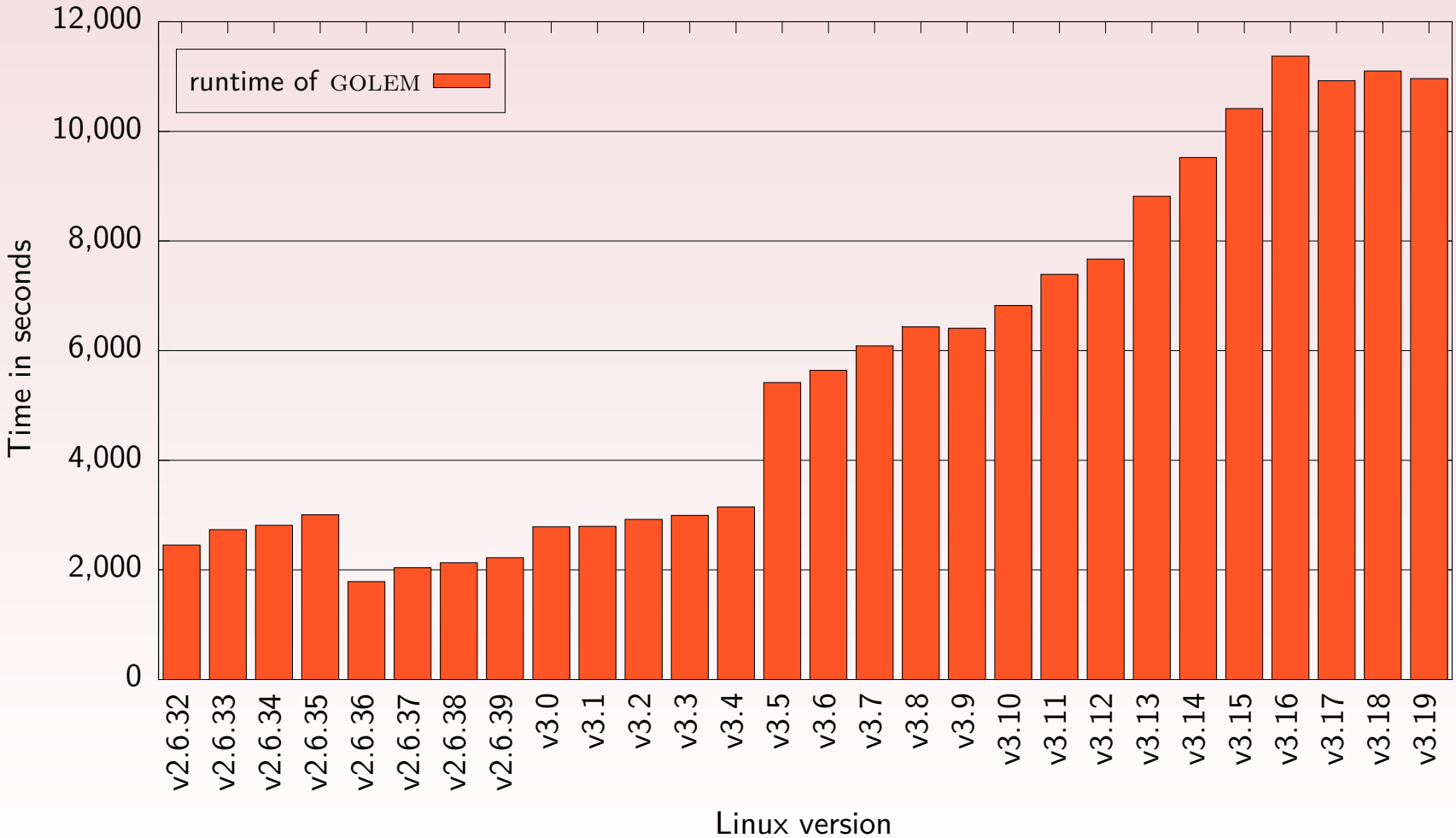
The UNDERTAKER Toolchain

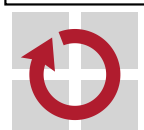


Recent Developments

- Extraction accuracy improvements for KCONFIG
- Code/Speed improvements (C++11, incremental SAT solving)
- UNDERTAKER-CHECKPATCH (Valentin Rothberg):
  - Analysis of patches submitted into the kernel
  - Comparison of the before and after states of the files changed by the currently checked patch
  - Improved reporting of newly introduced/fixed/unchanged defects
- Problem: KBUILD extractor, GOLEM, is very slow!

Recent Developments





Recent Developments

- Extraction accuracy improvements for KCONFIG
- Code/Speed improvements (C++11, incremental SAT solving)
- UNDERTAKER-CHECKPATCH (Valentin Rothberg):
  - Analysis of patches submitted into the kernel
  - Comparison of the before and after states of the files changed by the currently checked patch
  - Improved reporting of newly introduced/fixed/unchanged defects

- Problem: KBUILD extractor, GOLEM, is very slow!
- ⇒ Currently, no KBUILD data used in UNDERTAKER-CHECKPATCH

Table of Contents

Introduction

Fast KBUILD Data Extraction

Using the Data

Conclusion

How To Do It Fast?

- Dietrich (2012): Parsing (e.g., KbuildMiner) is not robust
  - across versions
  - regarding make language complexity

⇒ **probe** Kbuild and *infer* impact of options on file selection.

- But: Parsing is fast, while probing has become really slow

How To Do It Fast?

- Dietrich (2012): Parsing (e.g., KBUILDMINER) is not robust
  - across versions
  - regarding MAKE language complexity

⇒ **probe** KBUILD and *infer* impact of options on file selection.

- But: Parsing is fast, while probing has become really slow

- Idea:
  - Use **parsing**-based approach for the "simple" cases
  - Detect unparseable situation
  - Switch to more expensive, but possibly more resilient **probing** approach on demand.

How To Do It Fast?

- Dietrich (2012): Parsing (e.g., KBUILDMINER) is not robust
  - across versions
  - regarding MAKE language complexity

⇒ **probe** KBUILD and *infer* impact of options on file selection.

- But: Parsing is fast, while probing has become really slow

- Idea:
  - Use **parsing**-based approach for the "simple" cases
  - Detect unparseable situation
  - ~~Switch to more expensive, but possibly more resilient **probing** approach on demand.~~

- As it turns out: Parsing KBUILD can be fast, accurate and robust!

Parser Implementation

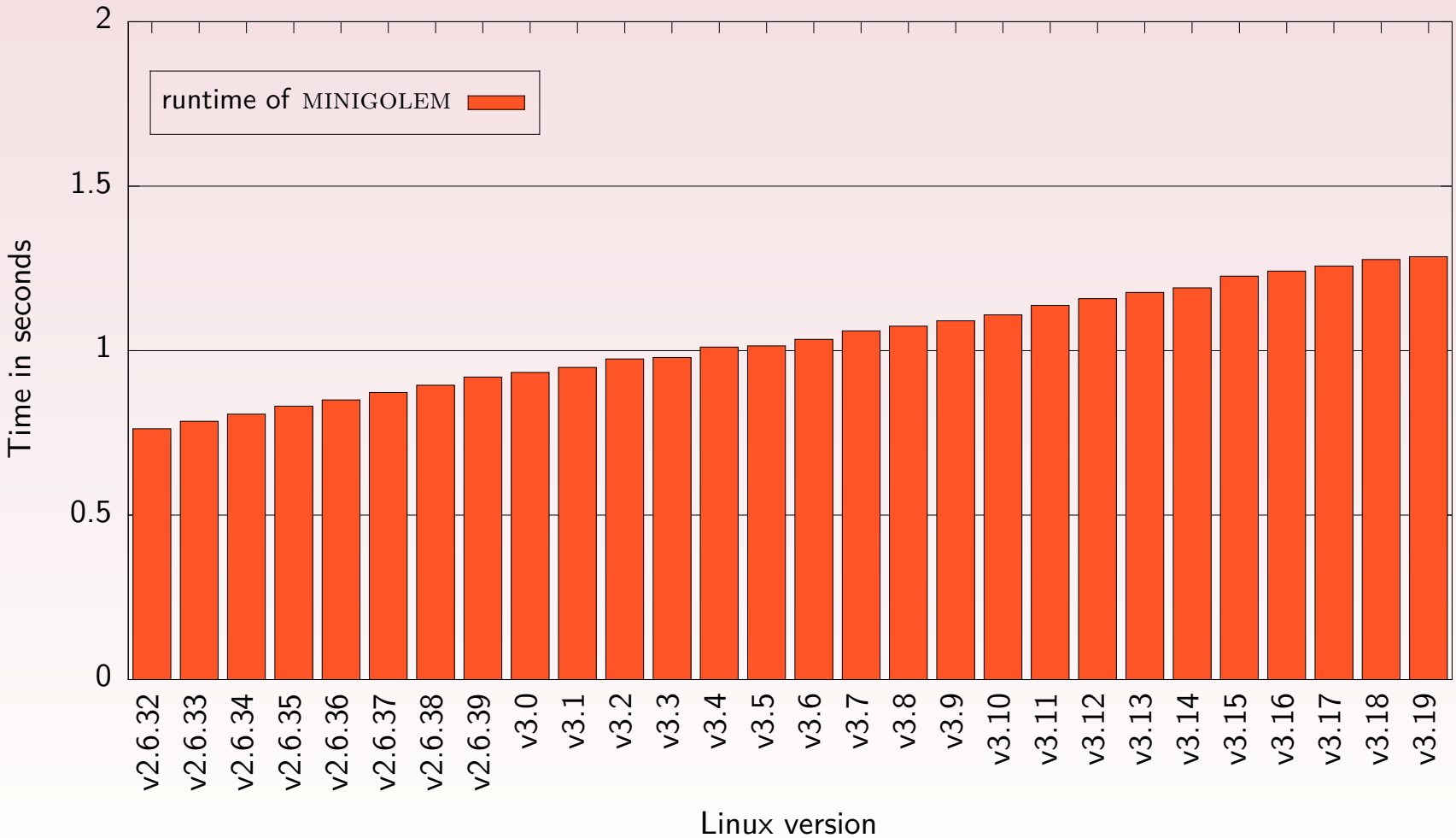
- I developed a modular parser, MINIGOLEM, in Python
- Core parser only processes files in generic way, project-specific "plug-in modules" implement actual extraction logic

⇒ Easy adaption for other projects (BUSYBOX, COREBOOT)

⇒ To treat additional special cases, only a small module has to be written instead of modifying existing code

⇒ Core parser: 192 LoC, Linux modules: 508 LoC

Parser Implementation



What about accuracy? (x86 architecture, v3.19)

	GOLEM	MINIGOLEM
Files found (total)	15,072 (96.1 %)	15,303 (97.6 %)
Files in both approaches	14,944	
⇒ Logically equivalent[1] PCs	14,831 (99.24 %)	

- **All** remaining formulas represent more accurate constraints in the parsing approach!
- 359 extra files found only by the parser
- Less than 90 files missing in MINIGOLEM, but present in GOLEM

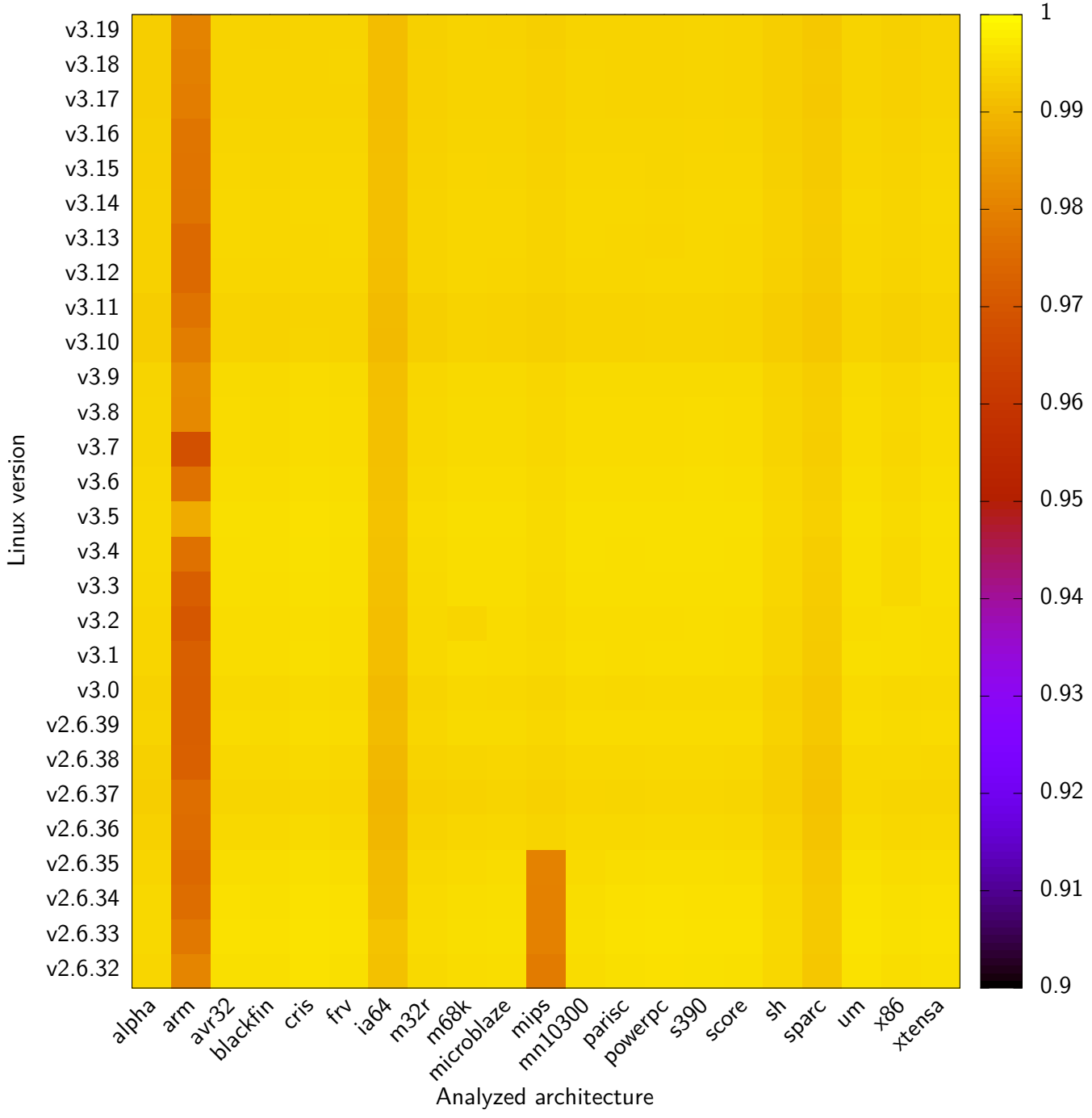
[1] Checked with LIMBOOLE (`http://fmv.jku.at/limboole`)

What about accuracy? (x86 architecture, v3.19)

	GOLEM	MINIGOLEM
Files found (total)	15,072 (96.1 %)	15,303 (97.6 %)
Files in both approaches	14,944	
⇒ Logically equivalent[1] PCs	14,831 (99.24 %)	

- **All** remaining formulas represent more accurate constraints in the parsing approach!
- 359 extra files found only by the parser
- Less than 90 files missing in MINIGOLEM, but present in GOLEM
- Not limited to this architecture/revision!

[1] Checked with LIMBOOLE (`http://fmv.jku.at/limboole`)

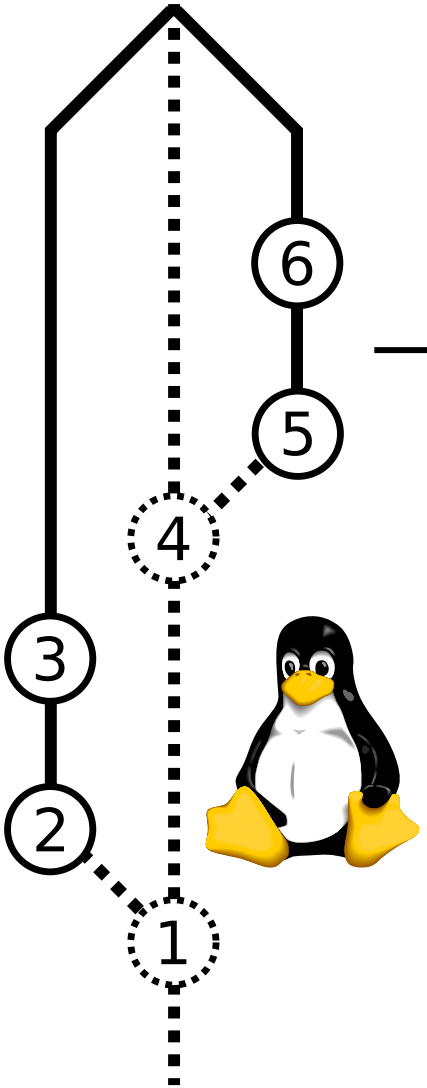


Percentage of logically equivalent presence conditions (GOLEM $\Leftrightarrow$ MINIGOLEM)

Daily Analysis of `linux-next`

- With the parser, overhead is small enough to also include KBUILD data into UNDERTAKER-CHECKPATCH
- Daily, incremental analysis of the `linux-next` development tree
- Detection of symbolic violations (i.e., reference to missing symbols) integrated into upstream `scripts/checkkconfigsymbols.py`
- ~50 defects reported and fixed (since January)

Daily Analysis of `linux-next`

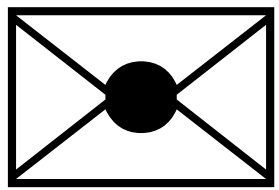


linux-next git tree

Defect analysis

```
    git reset --hard commit_1
undertaker-checkpatch -p commit_2
undertaker-checkpatch -p commit_3
    git reset --hard commit_4
undertaker-checkpatch -p commit_5
              ...
```

Defect reports



Notifications
to developers

Conclusion and Future Work

- Highly accurate extraction of variability data from KBUILD by parsing is feasible
- UNDERTAKER-CHECKPATCH can now take all layers of variability in Linux into account
- Daily analysis uncovers defects right when they are introduced

- How can we check more than just dead/undead `#ifdefs`?
- Can we use "something else" for remaining corner cases?

Conclusion

Questions?

andreas.ruprecht@fau.de