19th International Conference on Fundamental Approaches to Software Engineering (FASE)

FASE 2016

At ETAPS 2016: 2-8 April 2016, Eindhoven, The Netherlands http://www.etaps.org/index.php/2016/fase

FASE is concerned with the foundations on which software engineering is built. Submissions should make novel contributions to making software engineering a more mature and soundly-based discipline. Contributions should be supported by appropriate arguments and validation. Contributions that combine the development of conceptual and methodological advances with their formal foundations and tool support are particularly encouraged. We welcome contributions on all such fundamental approaches, including:

- Software engineering as an engineering discipline, including its interaction with and impact on society;
- Requirements engineering: capture, consistency, and change management of software requirements;
- Software architectures: description and analysis of the architecture of individual systems or classes of applications;
- Specification, design, and implementation of particular classes of systems: adaptive, collaborative, embedded, distributed, mobile, pervasive, or service-oriented applications

Abstracts due
9 October 2015 23:59 AoE (=GMT-12)
Papers due
16 October 2015 23:59 AoE (=GMT-12)
Author Notification
18 December 2015

Submission
15 pages + **2 extra pages** exclusively for references
Springer LNCS format
FASE'16 will not use a rebuttal phase

Keynote Speaker
Oscar Nierstrasz, Universität Bern, Switzerland
and other ETAPS invited speakers

Program Co-Chairs
Perdita Stevens, University of Edinburgh
Andrzej Wąsowski, IT University of Copenhagen

Program Committee
Sagar Chaki, Carnegie Mellon University
Nancy Day, University of Waterloo
Ewen Denney, NASA Ames
Juergen Dingel, Queen's University
Stéphane Ducasse, INRIA Lille - Nord Europe

Alexander Egyed, Johannes Kepler Universität Linz
Bernd Fischer, Stellenbosch University
Milos Gligoric, University of Illinois at Urbana-Champaign
Stefania Gnesi, ISTI-CNR
Valérie Issarny, INRIA Paris - Rocquencourt

Marta Kwiatkowska, University of Oxford
Barbara König, Universität Duisburg-Essen
Martin Leucker, Universität Lübeck

Hunting for Variability Bugs

Iago Abal
Aleksandar Dimovski
Claus Brabrand
Jean Melo
Andrzej Wąsowski

IT UNIVERSITY OF COPENHAGEN

PROCESS AND SYSTEM MODELS GROUP

Motivation

- Variability is out there
  - Allows to make systems more **adapatable**
  - Decreasing **cost** and **time-to-market**
  - Providing for **portability**
  - Allows massive **user side tailoring** (think highly configurable software systems)
- But variability brings a **cost**
  - Many recognize that managing **exponentially many software** variants is difficult.
  - Parsing **bugs**, typing bugs, linking bugs, and semantic bugs caused by variability are known
  - Many projects have started, including TypeChef, Undertaker, SPLLift, SNIP, Provelines, and Clafer ;)

- Variability bugs are horrible and nasty beasts
- Weapon #1: Variability abstractions
- Weapon #2: Effect-based abstractions

AGENDA

Let's have a look at a bug

index : kernel/git/stable/linux-stable.git

Linux kernel stable tree

summary refs log tree commit diff stats

author	Peter Hurley <peter@hurleysoftware.com>	2013-01-30 17:43:49 (GMT)
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2013-02-04 23:40:28 (GMT)
commit	7acf6cd80b201f77371a5374a786144153629be8 (patch)	
tree	5222e9eca68f3b37ad62d1eb74966705f12d1f96	
parent	16559ae48c76f1ceb970b9719dea62b77eb5d06b (diff)	

pty: Fix BUG()s when ptmx_open() errors out

```
If pmtx_open() fails to get a slave inode or fails the pty_open(),
the tty is released as part of the error cleanup. As evidenced by the
first BUG stacktrace below, pty_close() assumes that the linked pty has
a valid, initialized inode* stored in driver_data.

Also, as evidenced by the second BUG stacktrace below, pty_unix98_shutdown()
assumes that the master pty's driver_data has been initialized.

1) Fix the invalid assumption in pty_close().
2) Initialize driver_data immediately so proper devpts fs cleanup occurs.

Fixes this BUG:

[  815.868844] BUG: unable to handle kernel NULL pointer dereference at 0000000000000028
[  815.869018] IP: [<ffffffff81207bcc>] devpts_pty_kill+0x1c/0xa0
[  815.869190] PGD 7c775067 PUD 79deb067 PMD 0
[  815.869315] Oops: 0000 [#1] PREEMPT SMP
[  815.869443] Modules linked in: kvm_intel kvm snd_hda_intel snd_hda_codec snd_hwdep snd_pcm snd_seq_midi
```

See http://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/commit/?id=7acf6cd80b201f77371a5374a786144153629be8

The Variability Bugs Database Home Database Contribute search the database

Linux

Dereferencing uninitialized pointer causes Kernel crash

View raw files

During the initialization of a UNIX98 pseudo-terminal by ptmx_open, a tty_struct structure is allocated. But before its pointer field link->driver_data is properly initialized, ptmx_open will try to allocate an inode structure for the PTY slave. If this allocation fails, some cleanup code must be executed to free the already allocated resources. Namely, pty_close will be called to release the previously opened tty, and this eventually dereferences tty->link->driver_data, which is assumed to have been already initialized.

But fixed by commit 7acf6cd80b2
Parent commit tree here

Related links

Type	use of variable before initialization (CWE 457)
Config	UNIX98_PTYS && DEVPTS_MULTIPLE_INSTANCES (2nd degree)
C-features	FunctionPointers
Fix-in	code
Location	drivers/tty/

Simplified bug Simplified patch Single function bug Trace Discussion

See `http://vbdb.itu.dk/`, and add your own bugs

Let's have a look at a bug

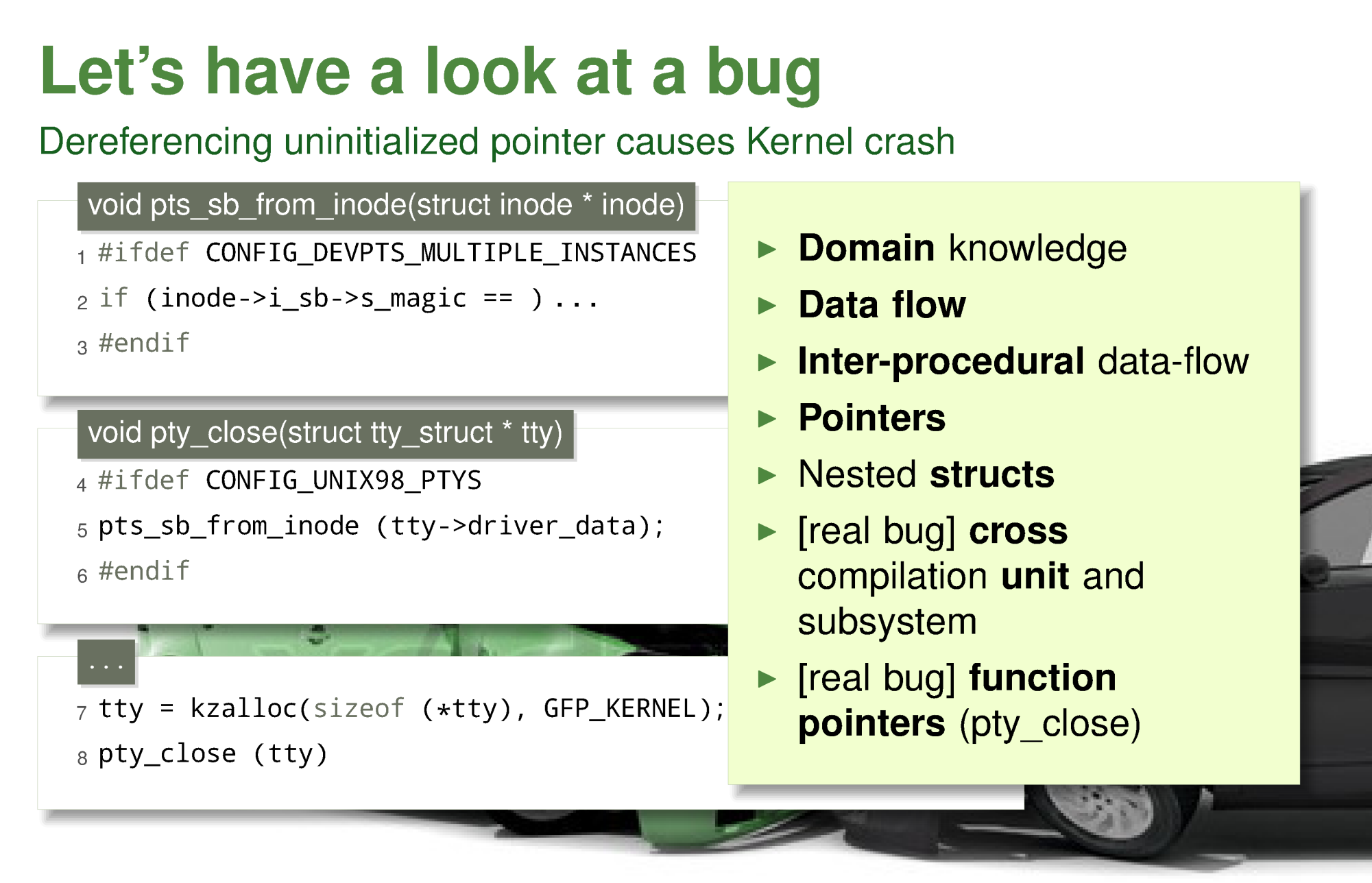
Dereferencing uninitialized pointer causes Kernel crash

```
void pts_sb_from_inode(struct inode * inode)
1 #ifdef CONFIG_DEVPTS_MULTIPLE_INSTANCES
2 if (inode->i_sb->s_magic == ) ...
3 #endif
```

```
void pty_close(struct tty_struct * tty)
4 #ifdef CONFIG_UNIX98_PTYS
5 pts_sb_from_inode (tty->driver_data);
6 #endif
```

```
...
7 tty = kzalloc(sizeof (*tty), GFP_KERNEL);
8 pty_close (tty)
```

- **Domain** knowledge
- **Data flow**
- **Inter-procedural** data-flow
- **Pointers**
- Nested **structs**
- [real bug] **cross** compilation **unit** and subsystem
- [real bug] **function pointers** (pty_close)



Bug 7acf6cd, see http://vbdb.itu.dk/#bug/linux/7acf6cd

Iago Abal, Claus Brabrand, Andrzej Wasowski. *42 variability bugs in the Linux kernel: a qualitative analysis.* ASE 2014: 421-432 + journal submission

Let's look at another bug

Control-flow

```
 1  extern int preempt_count;
 2
 3  void tcp_twsk_destructor() {
 4      #ifdef CONFIG_TCP_MD5SIG
 5      preempt_count--;
 6      #endif
 7  }
 8
 9  void inet_twdr_hangman(long dat
10      void (*fn)();              // fu
11      fn = (void (*)()) data; //
12      fn();                      // dyna
13  }
14
15  void __run_timers() {
16      long data = (long) &tcp_tws
17      int pc = preempt_count;
18      inet_twdr_hangman(data);
19      if (pc != preempt_count) BU
20  }
```

- Type **casts**, pointers to **ints**
- Do not loose **shapes**
- Unsafe casts help **generic programing**
- **Dynamic data structures** with pointers
- **Aliasing:** which pointers point to the same place
- Function pointers used heavily (**OO**)
- Inter-procedural data-flow not possible without **control-flow**
- [elsewhere] **conditional struct components** (with **incompatible casts**)

Bug 657e964e74


Variability Bug Hunting
Is For Tough Warriors,
Not for The Faint of Heart

- Variability bugs are horrible and nasty beasts
- Weapon #1: Variability abstractions
- Weapon #2: Effect-based abstractions

AGENDA

Variability bugs involve variability

Surprise, surprise!

- Linux has many configuration options
  - **5 000 — 15 000**, depending how you count
- Performance of lifted analyses depend on the **size of the configuration space**
- Many configs can be involved in a trace, but **many may be irrelevant** for a bug
- Bugs we found so far are up to **degree 5**, not 15 000

Programs are state transformers

Denotational semantics by example ...

For simplicty, we work with IMP, not with C ...

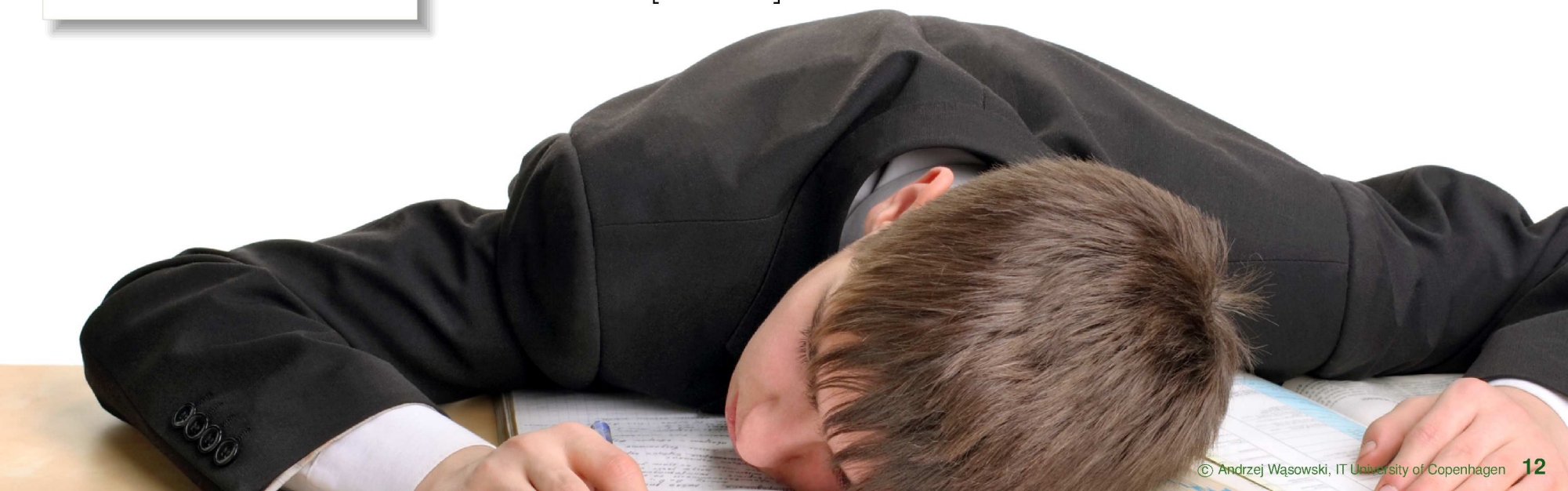
```
1 x := 0;
2 x := x + 1;
3 x := 1
```

$[x \mapsto ?\,]$

$[x \mapsto 0]$

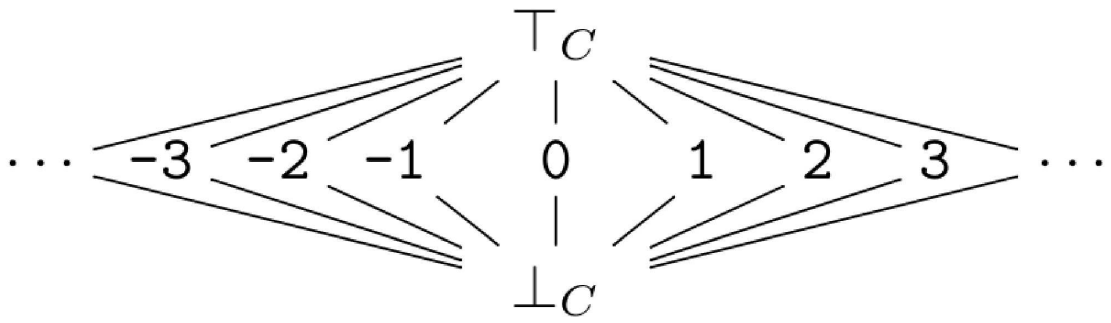
$[x \mapsto 1]$

$[x \mapsto 1]$



Executions transform domain elements

Example, in the previous slide: Constant Propagation Lattice C



- $[x \mapsto n]$ means that x has known **constant value**
- $[x \mapsto \top]$: x the value of x can have **any value** (value not known)
- $[x \mapsto \bot]$: x can have no value (inconsistency)
- Moving up in the lattice **decreases knowledge** (information)
- An execution on a more abstract domain is called an **analysis**
- Other abstract domains: signs, intervals, types, ...

Execution of a program with #ifdefs

Gives a **lifted** analysis

feature model

$$\varphi = A \vee B$$

variational program

```
1 x := 0;
2 #if A
3     x := x + 1;
4 #endif
5 #if B
6     x := 1
7 #endif
```

(symbolic) lifted stores

$$\text{true} \mapsto [x \mapsto \top]$$

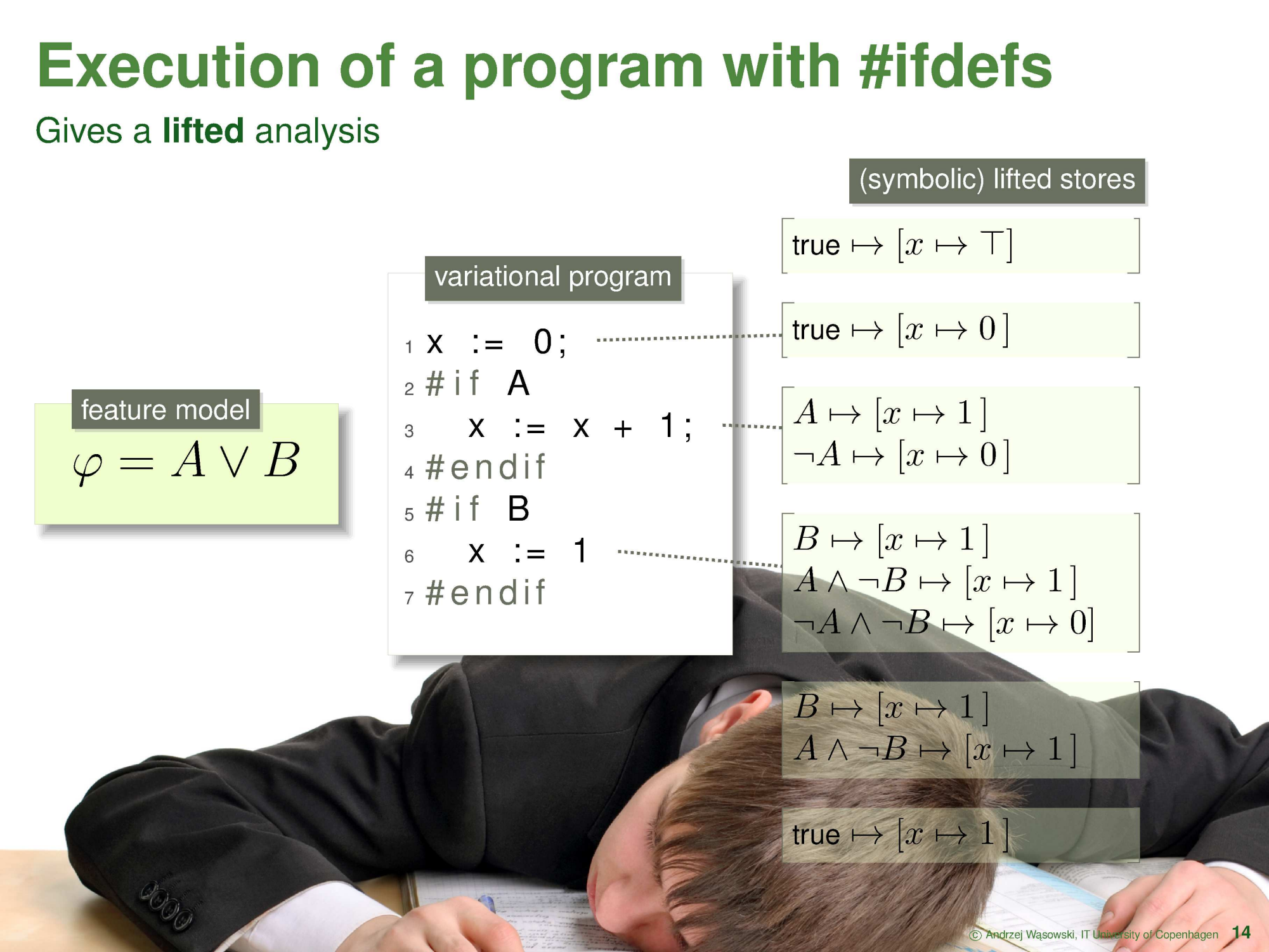
$$\text{true} \mapsto [x \mapsto 0]$$

$$A \mapsto [x \mapsto 1]$$
$$\neg A \mapsto [x \mapsto 0]$$

$$B \mapsto [x \mapsto 1]$$
$$A \wedge \neg B \mapsto [x \mapsto 1]$$
$$\neg A \wedge \neg B \mapsto [x \mapsto 0]$$

$$B \mapsto [x \mapsto 1]$$
$$A \wedge \neg B \mapsto [x \mapsto 1]$$

$$\text{true} \mapsto [x \mapsto 1]$$



“Join” abstraction confounds all variants

When you need a fast imprecise analysis, for instance in an IDE

feature model

$$\varphi = A \vee B$$

config-ns

A	B
0	1
1	0
1	1

$[x \mapsto 1]$

$[x \mapsto 1]$

$[x \mapsto 1]$

lub $\sqcup$

$[x \mapsto 1]$

$$\boldsymbol{\alpha}^{\text{join}}(\overline{a}) = \bigsqcup_{k \in \mathbb{K}} \pi_k(\overline{a})$$

$$\boldsymbol{\gamma}^{\text{join}}(a) = \prod_{k \in \mathbb{K}} a$$

config-ns

A	B
0	1
1	0
1	1

$[x \mapsto 1]$

$[x \mapsto 1]$

$[x \mapsto 0]$

lub $\sqcup$

$[x \mapsto \top]$

Variational abstract interpretation translates this to executions (cf. Claus’ talk)

J. Midtgaard. A. Dimovski. C. Brabrand. A. Wasowski. *Systematic Derivation of Correct Variability-Aware Program Analyses.* Science of Computer Programming 2015

“Project” ignores some variants

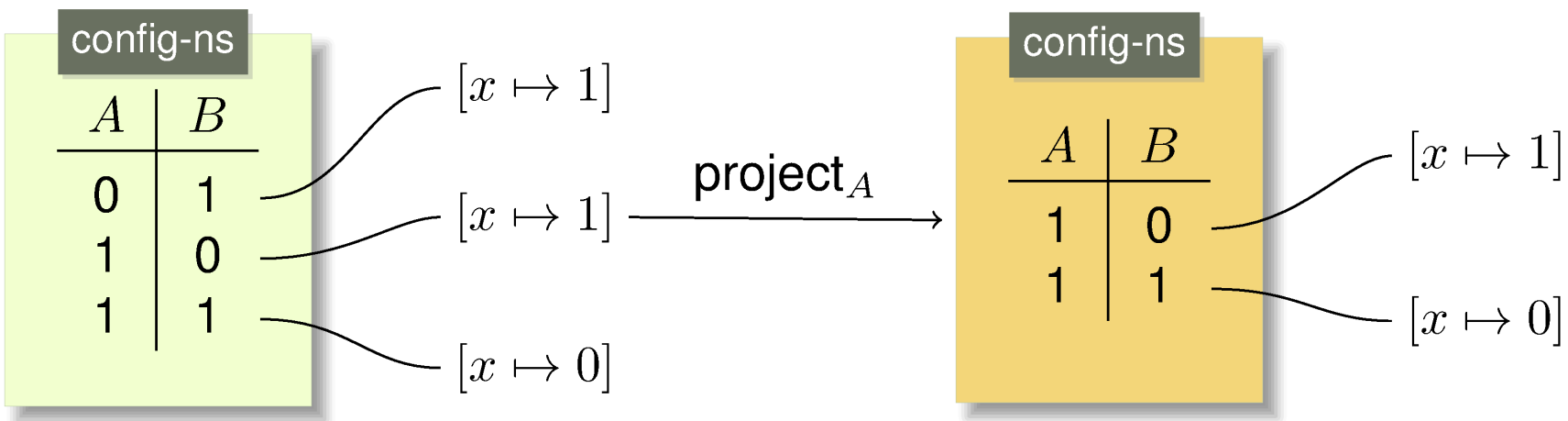
- **Sampling**
- Scale up using **divide & conquer**
- **Differentiate precision** for variants (say per SIL level)

feature model

$$\varphi = A \vee B$$

$$\alpha_\varphi^{\text{proj}}(\overline{a}) = \prod_{k \in \mathbb{K}, k \models \varphi} \pi_k(\overline{a})$$

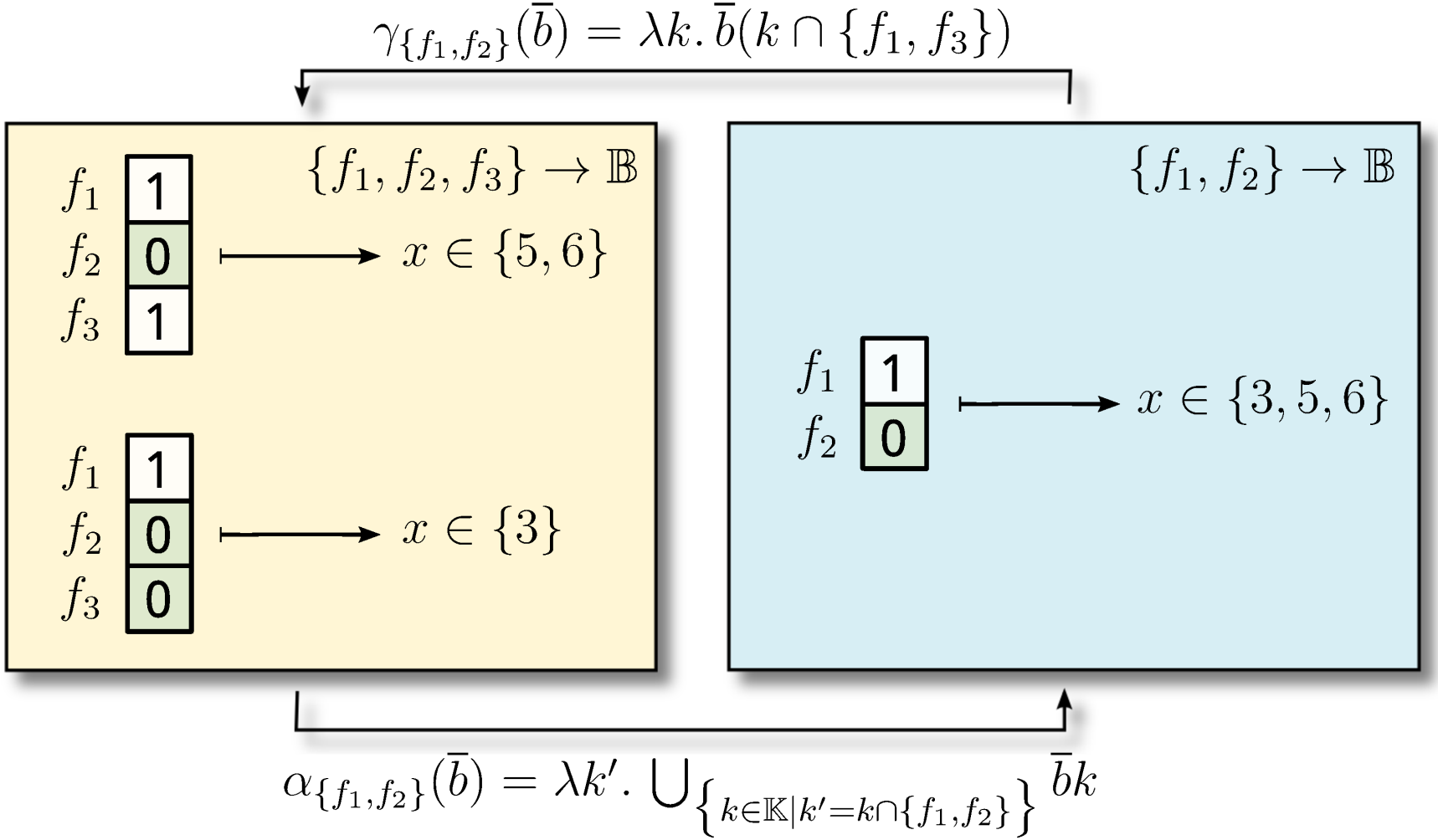
$$\gamma_\varphi^{\text{proj}}(\overline{a}') = \prod_{k \in \mathbb{K}} \begin{cases} \pi_k(\overline{a}') & \text{if } k \models \varphi \\ \top & \text{if } k \not\models \varphi \end{cases}$$



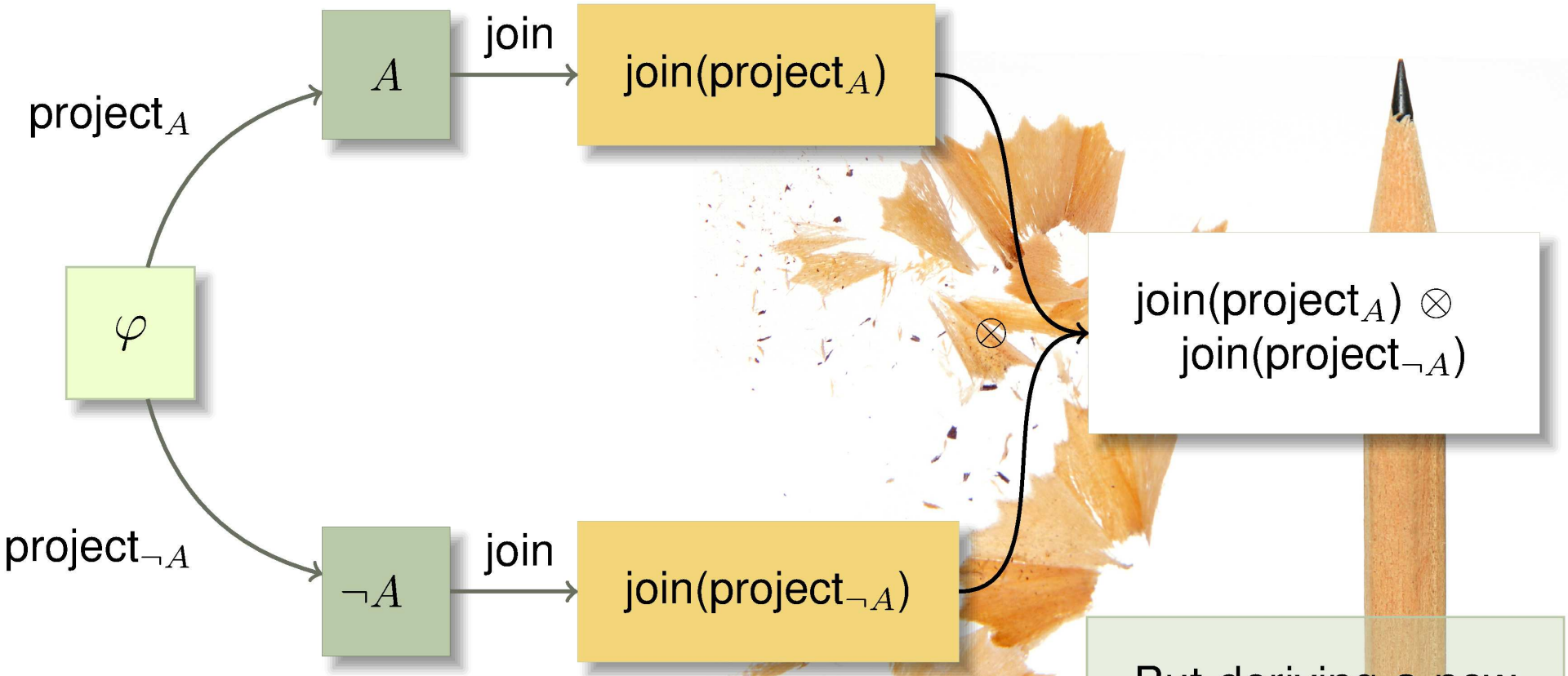
J. Midtgaard. A. Dimovski. C. Brabrand. A. Wasowski. *Systematic Derivation of Correct Variability-Aware Program Analyses.* Science of Computer Programming 2015

“Ignore” a feature

Inject domain knowledge to soundly help the analysis "go trough"



Composing Abstractions

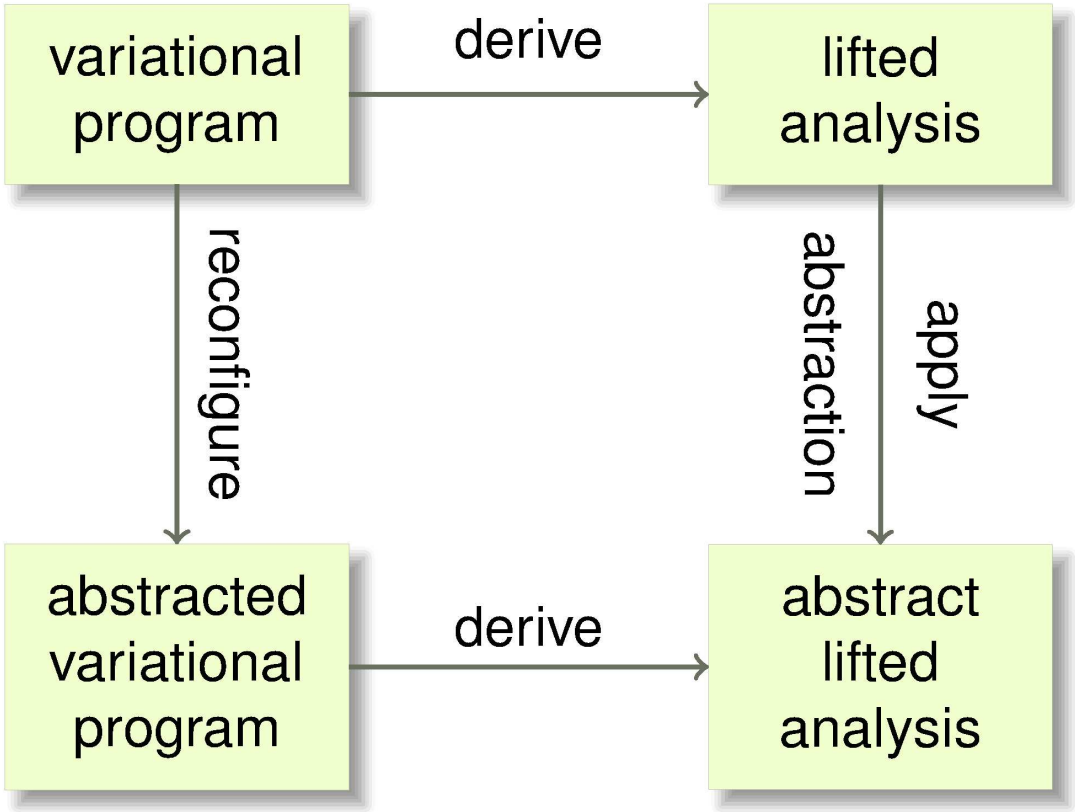


A calculus of variability abstractions

$$\alpha \;::=\; \boldsymbol{\alpha}^{\text{join}} \mid \boldsymbol{\alpha}_{\phi}^{\text{proj}} \mid \alpha \circ \alpha \mid \alpha \otimes \alpha$$

But deriving a new analysis for each new expression is very **laborious!**

Welcome to Reconfigurator



- One simplistic reconfigurator for **data-flow and Java**
- One simplistic reconfigurator for **model-checking and (f)Promela**

Aleksandar S. Dimovski, Claus Brabrand, Andrzej Wasowski. *Variability Abstractions: Trading Precision for Speed in Family-Based Analyses.* ECOOP 2015

Reconfigurator

Example

feature model

$$\varphi = A \vee B$$

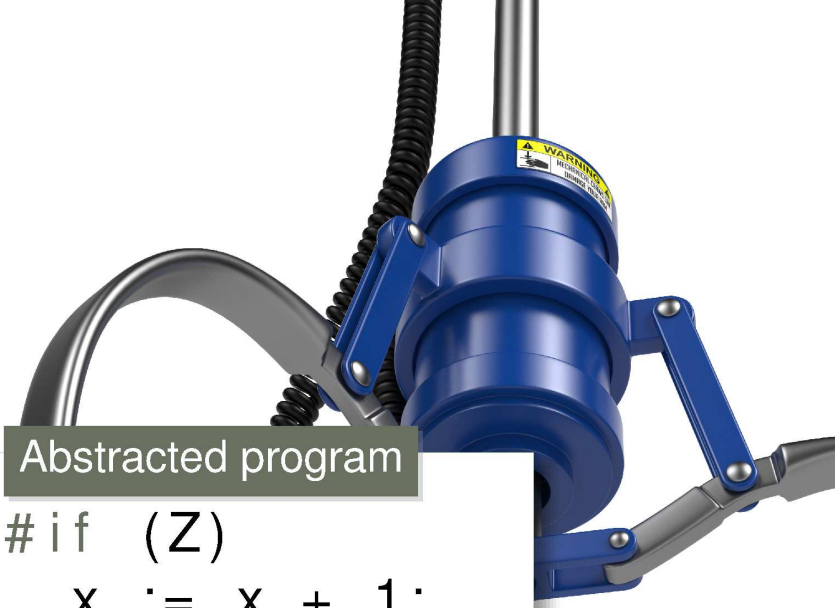
Input program

```
1 #if (A)
2     x := x + 1;
3 #endif
4 #if (B)
5     x := 1
6 #endif
```

$$\alpha_Z^{\text{join}} \circ \alpha_A^{\text{proj}}$$

Abstracted program

```
1 #if (Z)
2     x := x + 1;
3 #endif
4 #if (Z)
5     if (*)
6         x := 1
7     else
8         skip
9 #endif
```



- The outcome can be analyzed using **non-lifted scanner**
- Single product tools are **more mature** (handle more language!)

Reconfigurator

Performance evaluation

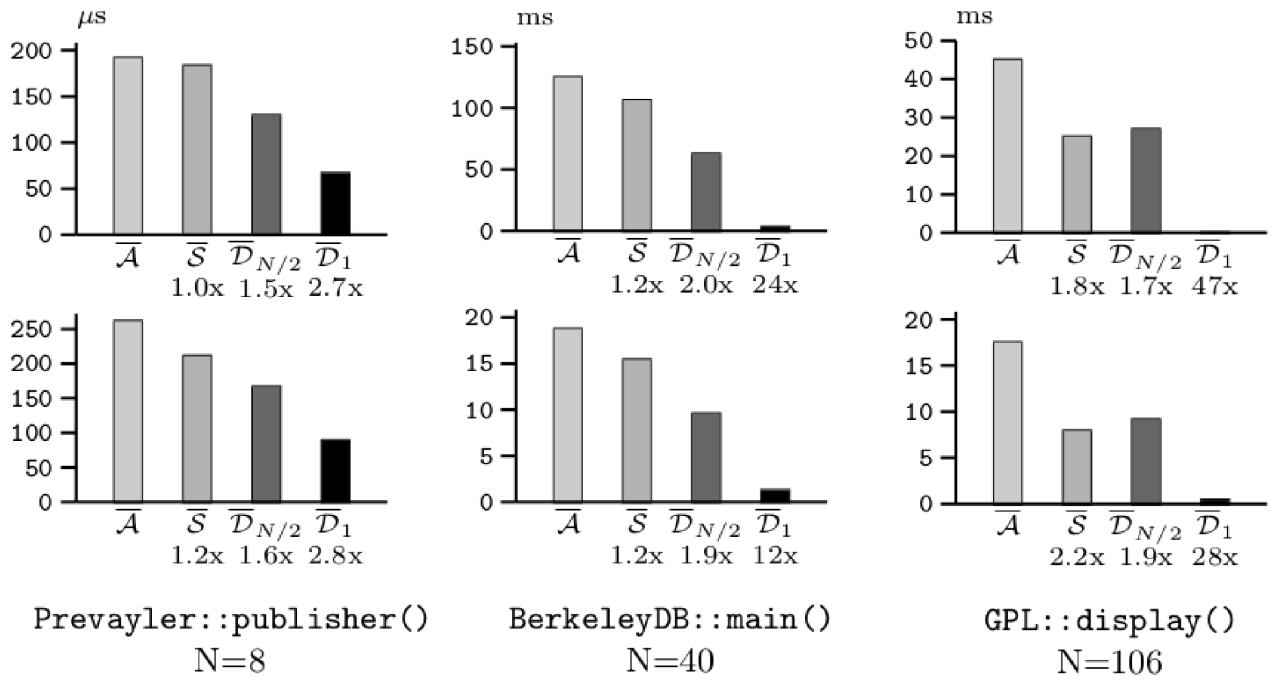


Fig. 9. Analysis time for *reaching definitions* (above) and *uninitialized variables* (below): $\overline{\mathcal{A}}$ (baseline) and $\overline{\mathcal{S}}$ (sharing) vs. $\overline{\mathcal{D}}_{N/2}$ (medium abstraction) and $\overline{\mathcal{D}}_1$ (maximum abstraction).

A. Dimovski. C. Brabrand, A. Wąsowski. *Variability Abstractions: Trading Precision for Speed in Family-Based Analyses.* ECOOP 2015

A. Dimovski, A. Al-Sibahi, C. Brabrand, A. Wasowski. *Family-Based Model Checking without a Family-Based Model Checker.* Under review.


Variability Bug Hunting
Is For Tough Warriors,
Not for The Faint of Heart

- Variability bugs are horrible and nasty beasts
- Weapon #1: Variability abstractions
- Weapon #2: Effect-based abstractions

AGENDA

Let's detect this error with Coccinelle

```
@@
type T;
T* x;
expression E;
@@

* x = kzalloc(...);
  ... when != (x = E)
      when != &x
* *x
```

- Coccinelle matches patterns over traces
- Inter-procedurally
- Supports CPP (improving thanks to Iago Abal)
- Efficient
- Integrated into the Linux kernel build system
- Does not know about aliasing

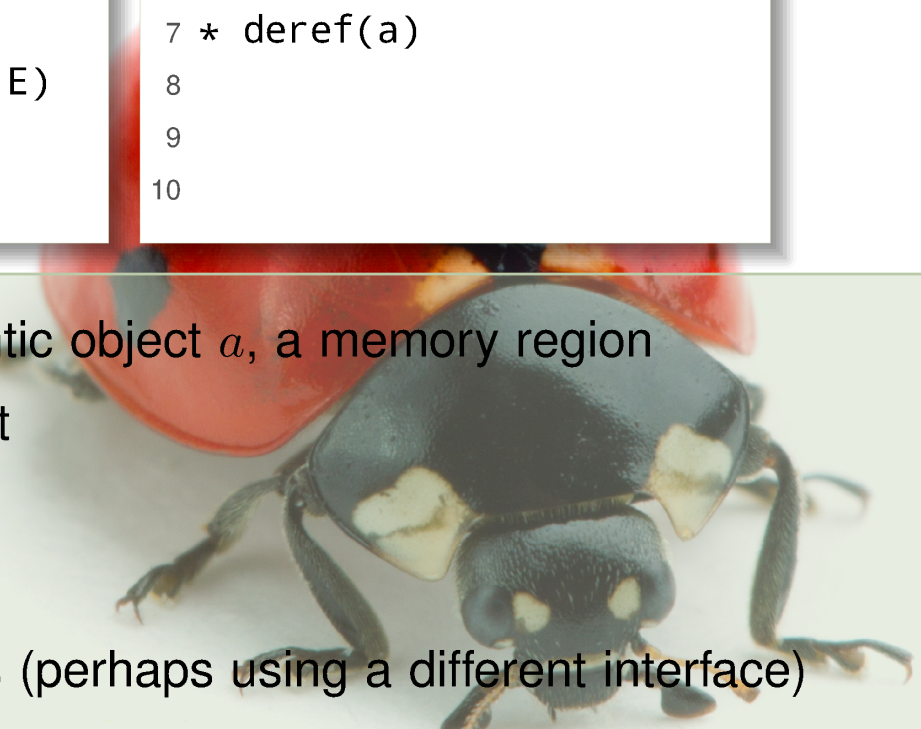
```
  size = sizeof(*new) + new_head_len + new_tail_len;
- new = kzalloc(size, GFP_KERNEL);
  if (!new)
          return -ENOMEM;
  [...]
- tty->head = ((u8 *) new) + sizeof(*new);
  tty->tail = new->head + new_head_len;
  tty->head_len = new_head_len;
  tty->tail_len = new_tail_len;
```

See http://coccinelle.lip6.fr/ by Julia Lawall and many others

A more semantic "Coccinelle"

```
1  @@
2  type T;
3  T* x;
4  expression E;
5  @@
6
7  * x = kzalloc(...);
8    ... when != (x = E)
9        when != &x
10 * *x
```

```
1  @@
2  pointer a;
3  @@
4
5  * a = kzalloc(...);
6    ... when != writes(a)
7  * deref(a)
8
9
10
```



- Declare a semantic object a, a memory region
- Track assignment
- No writes and
- A dereference
- Work in progress (perhaps using a different interface)
- To do this we need to know semantic properties of statements and expressions (such as writes(a) or deref(a))

Computing the semantic abstraction

Using a type & effect (shape & effect) system

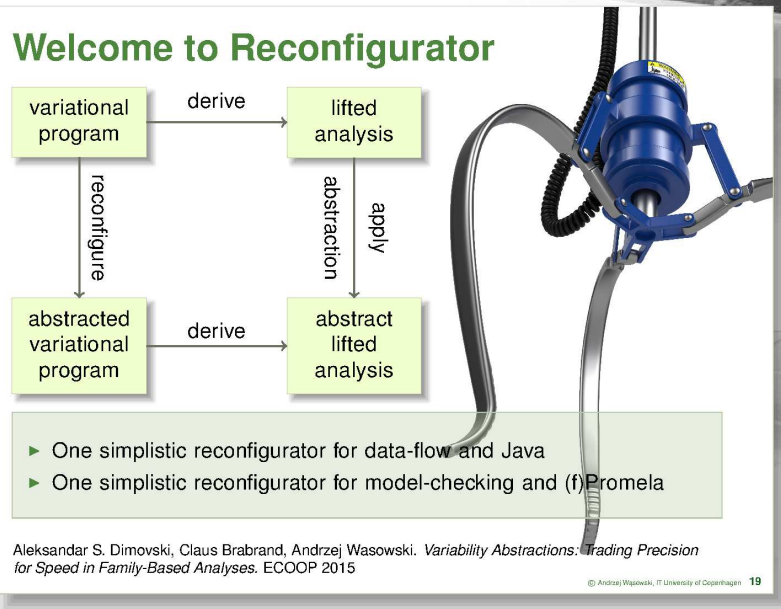
$$\textsc{Addr}\ \frac{\Gamma \vdash^{\mathrm{lval}} lv : T \ \&\ \mathtt{ref}_\rho\ \zeta\ \&\ \varphi \qquad \rho \neq \mathtt{reg}}{\Gamma \vdash \mathtt{\&}lv : T\mathtt{*}\ \&\ \mathtt{ptr}\ \mathtt{ref}_\rho\ \zeta\ \&\ \varphi}$$

$$\textsc{Lvalue-Exp}\ \frac{\Gamma \vdash^{\mathrm{lval}} lv : T \ \&\ \mathtt{ref}_\rho\ \zeta\ \&\ \varphi}{\Gamma \vdash lv : T\ \&\ \zeta\ \&\ \varphi \cup read_\rho}$$

- Shape: what is the memory structure at a given address
- Effect: reading, writing to a location, Linux specific effects
- 26 pages formal definition for a large part of C & ocaml implementation
- We can now type check most of simplified bugs from VBDB
- The type checker infers effects for each program point

Conclusion

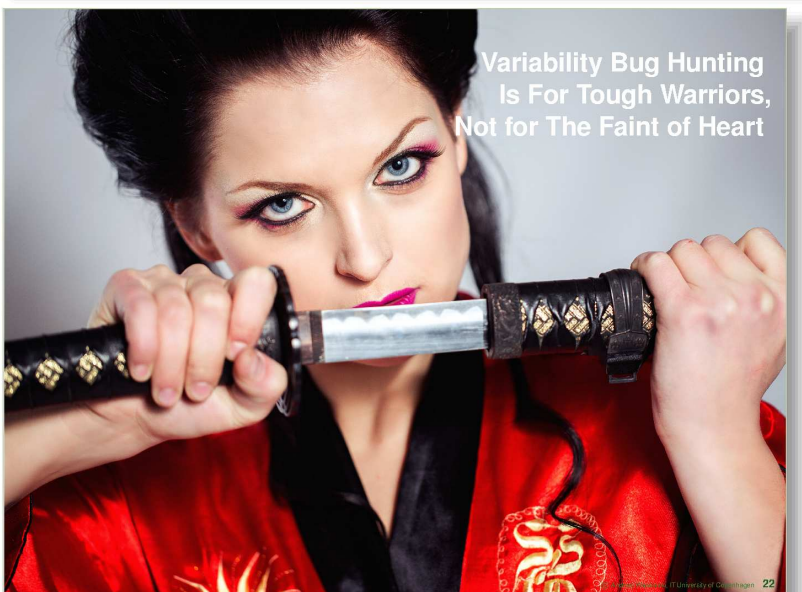
Welcome to Reconfigurator



- One simplistic reconfigurator for data-flow and Java
- One simplistic reconfigurator for model-checking and (f)Promela

Aleksandar S. Dimovski, Claus Brabrand, Andrzej Wasowski. *Variability Abstractions: Trading Precision for Speed in Family-Based Analyses.* ECOOP 2015

 19



Computing the semantic abstraction

Using a type & effect (shape & effect) system

$$\textsc{Addr}\ \frac{\Gamma \vdash^{\text{lval}} lv : T\ \&\ \texttt{ref}_\rho\ \zeta\ \&\ \varphi \qquad \rho \neq \texttt{reg}}{\Gamma \vdash \&lv : T\texttt{*}\ \&\ \texttt{ptr ref}_\rho\ \zeta\ \&\ \varphi}$$

$$\textsc{Lvalue-Exp}\ \frac{\Gamma \vdash^{\text{lval}} lv : T\ \&\ \texttt{ref}_\rho\ \zeta\ \&\ \varphi}{\Gamma \vdash lv : T\ \&\ \zeta\ \&\ \varphi \cup read_\rho}$$

- Shape: what is the memory structure at a given address
- Effect: reading, writing to a location, Linux specific effects
- 26 pages formal definition for a large part of C & ocaml implementation
- We can now type check most of simplified bugs from VBDB
- The type checker infers effects for each program point

 27