THALES

Prove your Colorings: Formal Verification of Cache Coloring of Bao Hypervisor

Axel Ferréol, Laurent Corbin, Nikolai KOSMATOV

Thales Research & Technology, cortAIx Labs,
Palaiseau (Paris-Saclay), France

EuroProofNet Symposium, Orsay, Sep 18, 2025
First presented: ETAPS / FASE 2025, Hamilton, May 06, 2025

Thales group Overview

over 77,000 employees

68 countries
Global presence

€1 bn
Self-funded R&D
* Does not include externally financed R&D

THALES

Thales' Mission

Sensing
& data gathering

Data transmission
& storage

Data processing
& decision making

Digital Identity and Security

Defence and Security

Aerospace

Space

We help customers master decisive moments by providing the right information at the right moment

Previous Work: Formal Verification of JavaCard Virtual Machine

- **Common Criteria: Int. Standard for Security Certification**
  - The highest level (EAL7) requires **a formal proof of correctness and security**
- **World-first formal verification of real-life smart card code for certification**
  - Previous certification approach with a high-level model not accepted anymore
- **EAL7 certificate issued by ANSSI in Oct. 2022**
- **Only 5th EAL7 certificate out of 1600+ certifications worldwide**
  - Competitors do not hold EAL7 certificates for smart cards today
- **Innovative methodology of formal verification for certification**

Publication: Adel Djoudi, Martin Hana and Nikolai Kosmatov.
"Formal verification of a JavaCard virtual machine with Frama-C". FM 2021.

Motivation and Main Goal of this Work

- **Hypervisors become highly relevant for critical embedded systems to enable more functions**
  - As it is not possible to add more hardware because of size, weight and cost constraints
- **Static hypervisors rely on partitioning of resources: each VM accesses its own resources**
- **Some resources must be shared, such as processor last-level cache (LLC)**
  - This cache is shared between several cores, each one possibly running a different VM
- **Cache coloring is used to split cache into several areas, each associated with a color**
- **Assigning different colors to VMs ensures isolation of the cache areas they use**
- **Cache coloring is complex and highly critical, its correctness is essential to guarantee**

- **Our goal: formal verification of the cache coloring in Bao, a static hypervisor**

Plan

- Frama-C verification platform
- Bao Hypervisor and Cache Coloring
- Identified Bugs and Corrections
- Verification Results and Statistics
- Verification Approach Highlights
- Conclusion and Future Work

Frama-C, its Deductive Verification plugin WP and Specification Language ACSL

Frama-C is a platform for analysis and verification of C programs

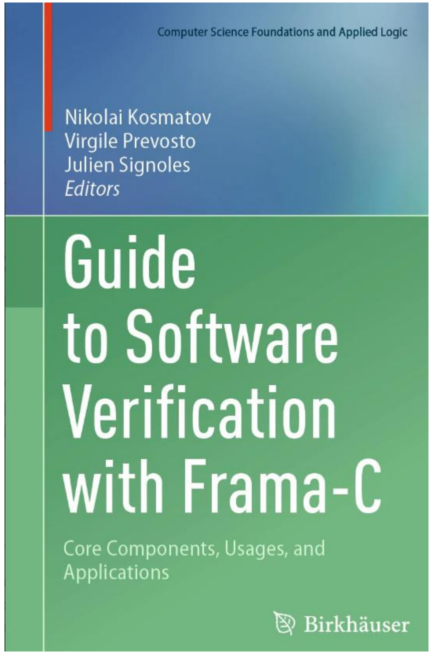
- It supports ACSL (ANSI C Specification Language)

Deductive verification plugin WP: based on weakest precondition

- Proof of semantic properties of the program
- Modular verification (function by function)
- Input: a program and its specification in ACSL
- WP generates verification conditions (VCs)
- Relies on Why3 and Automatic Provers to discharge VCs
  - Alt-Ergo, Z3, CVC5, …
- Interactive proof via WP proof scripts or in proof assistant Rocq

Value analysis plugin Eva: based on abstract interpretation

Software Analyzers


Computer Science Foundations and Applied Logic

Nikolai Kosmatov
Virgile Prevosto
Julien Signoles
Editors

Guide to Software Verification with Frama-C

Core Components, Usages, and Applications

Birkhäuser

Bao Hypervisor and Cache Coloring

- Bao [1] is a lightweight open-source static hypervisor for embedded systems
- Provides strong isolation between VMs and real-time guarantees
- Features an elegant implementation of cache coloring
  - A color is assigned to each memory page
  - Data of a memory page can be loaded only into cache sets of the same color
  - Only pages of certain color(s) can be assigned to each VM
  - Thus, a VM cannot access data of other VMs
- It is crucial to ensure correctness of this implementation

[1] https://github.com/bao-project/bao-hypervisor

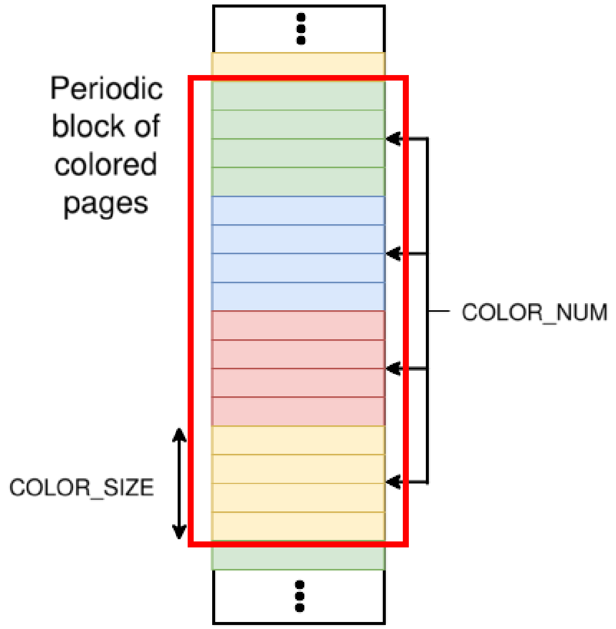
THALES

Implementation: Example of Page Coloring

The main memory layout in Bao with activated coloring looks like this:

- Pages are colored into the same color by consecutive groups of COLOR_SIZE pages
- Colors follow a constant sequence that loops every COLOR_NUM groups
- The periodic block of colors is repeated to engender the coloring of pages for the whole memory

- In this example, we consider 4 colors and 4 consecutive pages in each group of the same color



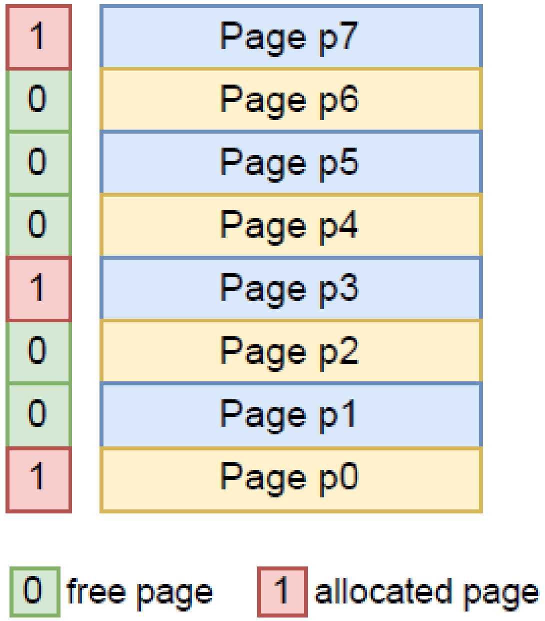
Implementation: Looking for n Free Consecutive Suitably-colored Pages (pset)

Allocation of n pages to a VM requires to find **a set of n free consecutive suitably-colored pages** inside a given pool of pages. We call such a set a **pset.**

If we **know the first page,** we **can recover all pages of the set.**

Examples for the figure nearby:

- {p2, p4, p6} a pset of size 3 for the yellow color? **Yes**
- {p1, p3} a pset of size 2 for the blue color? **No, p3 not free**
- {p1, p5} a pset of size 2 for the blue color? **No, not consecutive, p3 is blue and in-between**

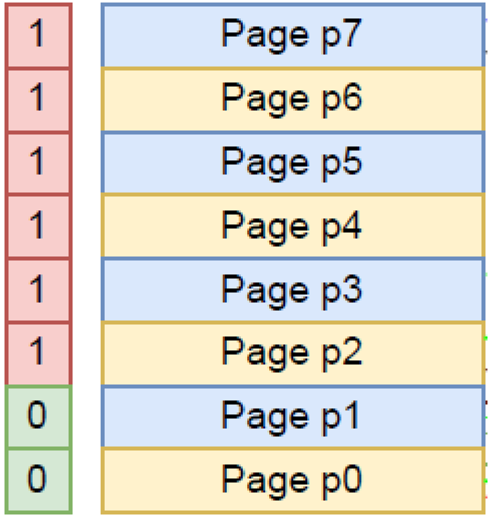


Identified Bugs: First Example

Allocation of a set of 2 free consecutive blue pages
when the starting page is p7
returns a wrong result: the set {p1, p3}
instead of returning false (not found)

Page p3 has already been allocated, possibly to the same VM or another VM accepting the blue color!

We confirmed this counterexample with the value analysis plugin Eva of Frama-C.



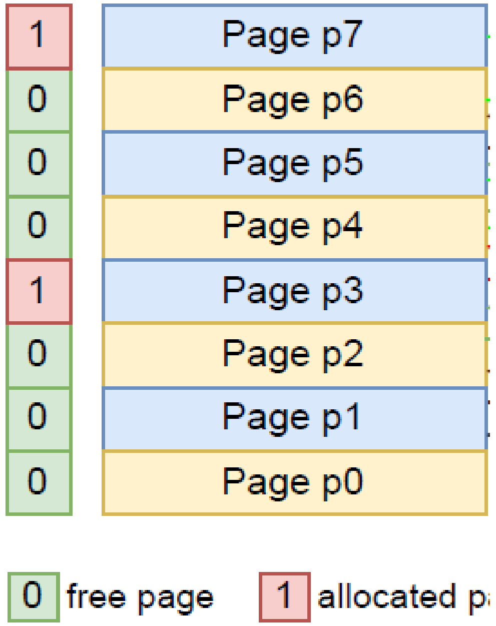
Identified Bugs: Second Example

Allocation of a set of 2 free consecutive blue pages
when the starting page is p1
returns a wrong result: the set {p5, p7}
instead of returning false (not found)

Page p7 has already been allocated, possibly to the same VM or another VM accepting the blue color!

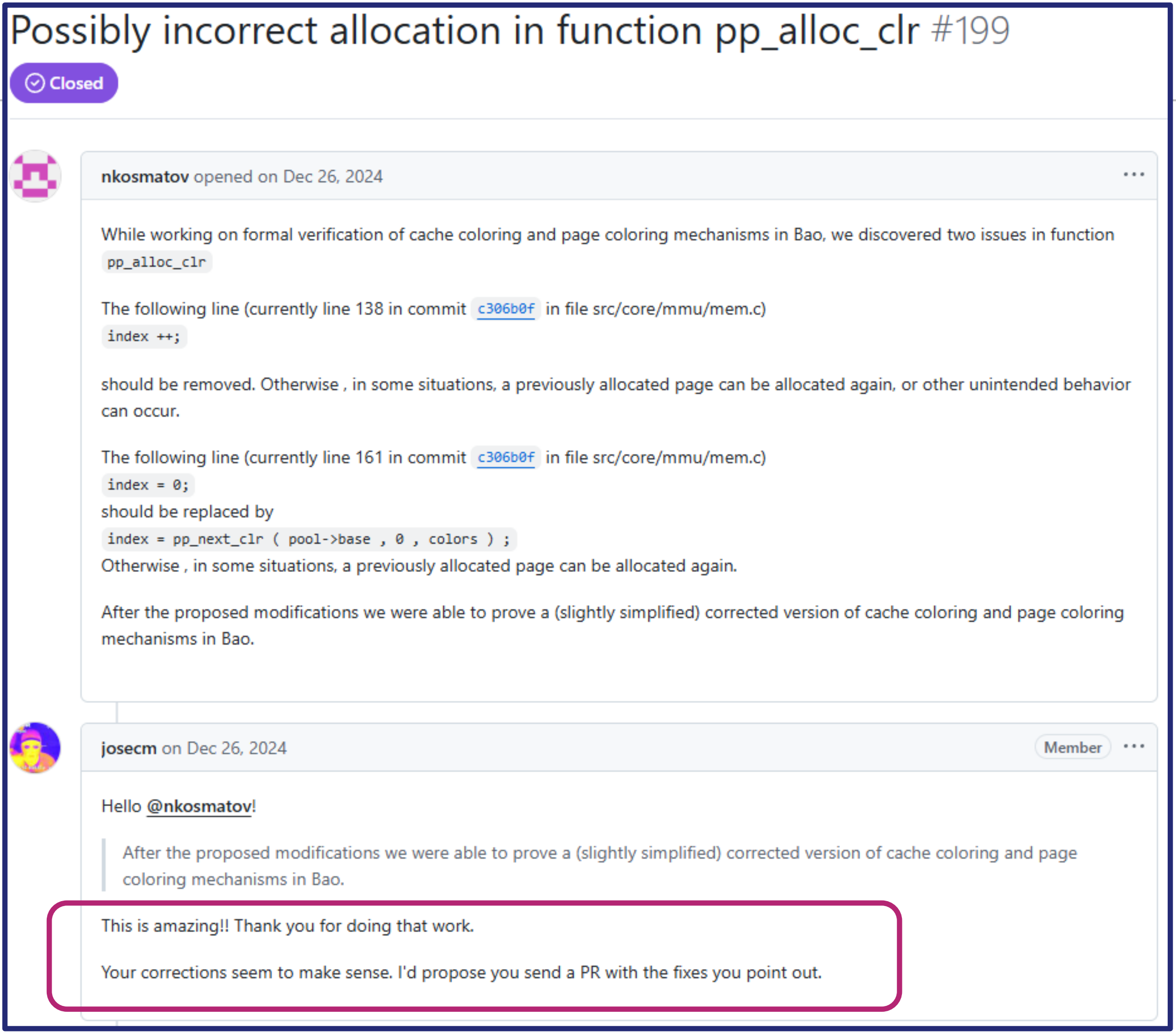
In a variant, page p7 may be non-existing, or have already been allocated, possibly to the hypervisor, same VM or another VM accepting the blue color!

We confirmed this counterexample with the value analysis plugin Eva of Frama-C.



Issue Reported and Fixed:

- We reported the issue to Bao developers and proposed bug fixes
- Our corrections were integrated
- Bugs were present since 2020!
- We proposed further code optimizations in the paper



Possibly incorrect allocation in function pp_alloc_clr #199

Closed

nkosmatov opened on Dec 26, 2024

While working on formal verification of cache coloring and page coloring mechanisms in Bao, we discovered two issues in function `pp_alloc_clr`

The following line (currently line 138 in commit `c306b0f` in file src/core/mmu/mem.c)
`index ++;`

should be removed. Otherwise , in some situations, a previously allocated page can be allocated again, or other unintended behavior can occur.

The following line (currently line 161 in commit `c306b0f` in file src/core/mmu/mem.c)
`index = 0;`
should be replaced by
`index = pp_next_clr ( pool->base , 0 , colors ) ;`
Otherwise , in some situations, a previously allocated page can be allocated again.

After the proposed modifications we were able to prove a (slightly simplified) corrected version of cache coloring and page coloring mechanisms in Bao.

josecm on Dec 26, 2024 Member

Hello @nkosmatov!

> After the proposed modifications we were able to prove a (slightly simplified) corrected version of cache coloring and page coloring mechanisms in Bao.

This is amazing!! Thank you for doing that work.

Your corrections seem to make sense. I'd propose you send a PR with the fixes you point out.

Verification Results and Statistics

- **Formal verification for 4 key functions: pp_next_clr, bitmap_get, bitmap_set, pp_alloc_clr**
  - To check consistency, also for 2 (simplified) upper-level functions: mem_alloc_ppages and mem_map
- **Stats: 100 lines of C code, 600 lines of ACSL annotations**
  - Slightly simplified C code, but all semantic behavior in the key functions is preserved
  - Annotations include function and loop contracts, predicates, ghost code, assertions, lemmas
- **The full proof with Frama-C/WP v. 29.0 takes ~5 min**
  - with options -wp-par=8 and -wp-timeout=40
- **523 proof goals**
  - 465 (88%) are proved automatically (by Frama-C, internal solver QED or SMT solvers Alt-Ergo, Z3, CVC5)
  - 55 (11%) proved by interactive WP proof scripts
  - 3 (1%) in proof assistant Rocq

THALES

Verification Highlights: Arithmetic Lemmas Proved in Rocq

- Page colors are efficiently computed with modulo and division operations
- Solvers are unable to handle such operations directly
- To address this, we use 4 arithmetic lemmas, and prove 3 of them in Rocq
- For instance, the next lemma gives an optimized way to compute the color of a page:

```
lemma arith_1: ∀ ℤ a,b,c,d; 0 ≤ a ∧ 0 ≤ b ∧ 0 < c ∧ 0 < d ⇒
  ((a+b)/c)%d == ((a+b%(c*d))/c)%d;
```

THALES

Verification Highlights: Ghost Arrays, 1/2

Ghost arrays help solvers reason on bits:

- We maintain an equivalence between a bitvector of colors and a **ghost array of integers**
- Such **flattening invariants** facilitate reasoning (on integers in array cells instead of bits)

THALES

Verification Highlights: Ghost Arrays, 2/2

Ghost arrays used to specify the existence case:

- The existence of a required pset is expressed with a witness, a global ghost array
- This simple solution avoids more complex alternatives with lots of quantifiers when a witness has to be found inside the function from an existential precondition
- We prove that if a witness pset exists, the allocation function finds a pset
- Specification completeness also requires the proof: if a pset does not exist, it is not found
- Instead, we prove: if a pset is found, it was initially present

Ghost arrays help to prove termination:

- Maximal number of loop steps is computed in ghost code using a witness

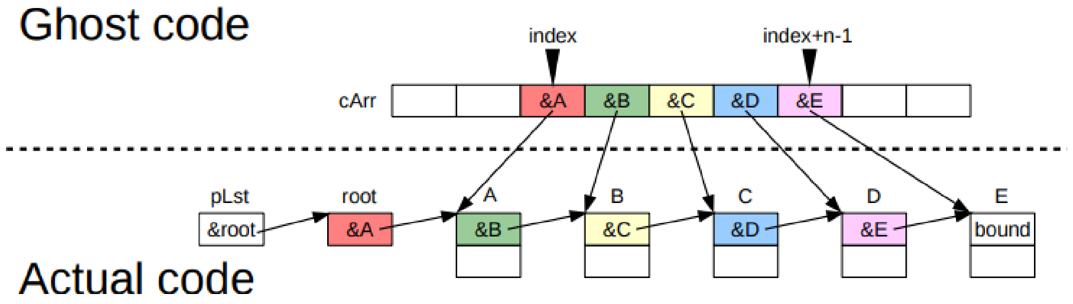
THALES

Verification Highlights: Linked Lists (and Ghost Arrays Again)

- A difficulty to handle linked lists used to represent pools of pages
- List nodes contain several data fields and pointers to external arrays

We use a technique inspired by previous work [Blanchard et al, NFM 2018, SAC 2019]:

- introduce a companion ghost array containing the addresses of the nodes of the linked list
- define and maintain a suitable linking predicate, which establishes the link between them



Verification Highlights: Non-trivial Invariants for Nested Loops

- With three levels of nested loops, carefully chosen loop invariants were necessary

- To prove that the function finds a pset when there exists a suitable pset in the pool, we have to ensure that if such a pset exists in the pool, then it is not missed
  - it is either located in the part of the pool the function has not explored so far, or
  - the function is about to find it, that is, the candidate pset is (partly) inside the witness

- We carefully express this property in the nested loops to have it preserved

- Notice that the function can possibly find another pset before the witness if it exists

Verification Highlights: Separation Issues and WP scripts

- **Additional separation clauses were often necessary**
- **Difficulties in proving preservation of seemingly trivial properties through assignments**
  - In the memory model of WP, pointers are treated as indices within arrays, where cells correspond to the pointed values
  - Consequently, properties involving pointers are translated into properties over arrays in WP
  - When a pointed value is modified, the whole array is seen as possibly modified, making proofs non-trivial for solvers
- **Manually created proof scripts in WP to show that values in predicates remain unchanged**

- **This leads to a significant verification overhead making the proofs complex to maintain**
- **A proof script stability issue in WP was detected and reported**

THALES

Companion Artifact

- **A companion artifact is available**
- **It can be used to reproduce the proof**
- **It contains**
  - All tools installed in a ready-to-use VM
  - Annotated code of the case study
  - WP proof scripts and proofs in Rocq
  - Counterexamples that can be used to confirm the bugs in the value analysis plugin Eva of Frama-C

Conclusion and Future Work

- Successful formal verification of cache coloring in Bao with Frama-C
- An original, industrially-relevant and security-critical target
  - The code is very elegant but challenging for deductive verification: it contains bit-level operations, non-trivial logic, complex arithmetic operations, multiple nested loops, linked lists
- We give its pedagogical presentation and emphasize main aspects of verification
- Two subtle bugs in the target code were identified and fixed, further optimizations were suggested, a minor issue in the verification tool was reported

Future work includes

- verification of optimized versions of cache coloring
- a larger verification of critical parts of the Bao hypervisor
- long-term goal: a highly optimized, verified static hypervisor for embedded systems