Mieux automatiser la vérification déductive avec des stratégies de preuve dans Frama-C/WP

Initially présented at TACAS 2024: Automate where Automation Fails: Proof Strategies for Frama-C/WP

Loïc CORRENSON, Allan BLANCHARD (CEA List),
Nikolai KOSMATOV, Adel DJOUDI (Thales)

AFADL 2024, Strasbourg, le 5 juin 2024

OPEN

Tool context: ACSL, Frama-C and its deductive verification plugin WP

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

- **ACSL (ANSI C Specification Language)** supported by Frama-C

Software Analyzers

WP plugin: Weakest Precondition based tool for deductive verification

- **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 Theorem Provers** to discharge VCs
  - **Alt-Ergo**, Z3, CVC4, CVC5, …

THALES

Example of a C program annotated in ACSL

```
/*@ requires n>=0 && \valid(t+(0..n−1));
      assigns \nothing;
      ensures \result != 0 <==>
        (\forall integer j; 0 <= j < n ==> t[j] == 0);
*/
int all_zeros(int t[], int n) {
  int k;
  /*@ loop invariant 0 <= k <= n;
        loop invariant \forall integer j; 0<=j<k ==> t[j]==0;
        loop assigns k;
        loop variant n−k;
  */
  for(k = 0; k < n; k++)
    if (t[k] != 0)
      return 0;
  return 1;
}
```

Can be proven with Frama-C/WP

THALES

Specification and Verification of Global Properties (Metaproperties) with MetAcsl

Resulting code after generating assertions with MetAcsl and proof with Frama-C/WP:

```
/*@ meta "A_unchanged_unless";
 */
/*@ requires
    ensures
      (C ≥ 0
      (C < 0
    assigns A,
  */
void foo(voi
{
  (C >= 0) {
    /*@ check A_unchanged_unless: _1: meta: C < 0 → \separated(&A, &A);
    A = C;
    /*@ check A_unchanged_unless: _2: meta: C < 0 → \separated(&B, &A);
    B = C;
  }
  return;
}
```

If all instances are proved, the metaproperty is true

MetAcsl

Contrary to an assert, a check is not kept in the proof context and does not overload the proof

MetAcsl instantiates a metaproperty in all relevant locations

Initial C code:

test5.c

```
int A, B, C;
/*@
  meta \prop, \name(A_unchanged_unless),
     \targets(\ALL), \context(\writing),
     C < 0 ==> \separated(\written, &A);
*/
/*@
  requires A==B;
  assigns A,B;
  ensures C>=0 && A==C && B==C ||
    C<0 && A==\old(A) && B==\old(B); */
void foo(){
  if ( C >= 0 ){
    A = C;
    B = C;
  }
```

Motivation: avoid interactive proof

- **Many successful applications of deductive verification in recent years**
- **Deductive verifiers manage to automatically prove the greatest number of proof goals, also called proof obligations, or verification conditions (VCs)**
  - This is in particular due to powerful and constantly evolving SMT solvers they rely on.
- **The remaining unproven goals typically require some form of interactive proof:**
  - with a proof script indicating a few initial proof steps to make the goal more suitable for an automatic prover, or
  - a fully interactive proof in a proof assistant like Coq.
- **The need for an interactive proof remains an important obstacle to a wider application of deductive verification on large projects**

THALES

Proposal: New proof strategy mechanism to generate scripts automatically

Instead of creating the proof script interactively in Frama-C/WP

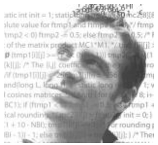
- **With much time** spent for try-and-wait-and-debug attempts

The verification engineer creates a proof strategy

- Written directly **in the source code** as a special annotation
- Including one or **several alternatives** (proof tactics) to try
  - unfolding, rewriting, enumerating, calling a solver,…
- Indicating possible strategies to apply on the **resulting proof goals** (children)
- Possibly attached to **specific proof goals**
- Typically, applied to **help automatic SMT solvers** to prove the goal

The tool automatically tries to apply provided strategies and records a proof script when the proof succeeds

Patrick Baudin, François Bobot, Loïc Correnson, Zaynah Dargaye, Allan Blanchard



2.5 Proof Strategies

Introduced since Frama-C 28.0 (Nickel)

Proof obligations generated by WP are usually discharged by an SMT solver specified by the user through command line option `-wp-prover`. As described in previous sections, complex proof obligations may also be split into simpler sub-goals by applying *Tactics* from the TIP user interface.

Proof strategies provide user-defined heuristics to automatically try various combinations of provers, timeouts and tactics, depending on the proof context. This is a much more effective technique than relying on manually edited scripts through the TIP user interface. Here are some benefits of using proof strategies:

- Proof strategies are automatic: there is no need for entering GUI session.
- Proof strategies can be associated to individual functions, lemmas or properties, or tried globally.
- Tactics are applied following patterns: depending on your case study, you can define fine-tuned strategies to solve your common issues.

Strategy — Alternatives

$$
\begin{array}{rl}
\multicolumn{2}{l}{\mathit{alternative} ::=} \\
| & \texttt{\char92prover("p",...,"p",}\ \mathit{timeout}\texttt{)} \\
| & \texttt{\char92tactic("id",}\ \mathit{param}, \ldots, \mathit{param}\texttt{)} \\
| & \texttt{\char92auto("id",...,"id")} \\
| & \texttt{\char92default} \\
| & \mathit{strategy}
\end{array}
$$

New proof strategy mechanism : examples

Example 1: a lemma unproven in Frama-C/WP with Alt-Ergo

```
lemma vhm_preserved{L1,L2}:
  valid_heap_model{L1} ∧
  mem_model_footprint_intact{L1,L2} ∧
  \at(gNumObjs,L1) == \at(gNumObjs,L2) ∧
  object_headers_intact{L1,L2}
  ⇒ valid_heap_model{L2};
```

A proof strategy that generates a script proving the lemma

```
strategy FastAltErgo:  \prover("alt-ergo", 1); // run Alt-Ergo for 1s
strategy EagerAltErgo: \prover("alt-ergo",10); // run Alt-Ergo for 10s
strategy UnfoldVhmThenProver:                  // Strategy with three steps:
  FastAltErgo,                                 // 1) fast prover attempt
  \tactic("Wp.unfold",                         // 2) if unproved, unfold
    \pattern(P_valid_heap_model((..))),        // predicate valid_heap_model
    \children(UnfoldVhmThenProver) ),          // and apply itself recursively
  EagerAltErgo;                                // 3) longer prover attempt
proof UnfoldVhmThenProver: vhm_preserved;      // Associate strategy to goal
```


THALES

New proof strategy mechanism : examples

Example 2: a lemma unproven in Frama-C/WP with Alt-Ergo

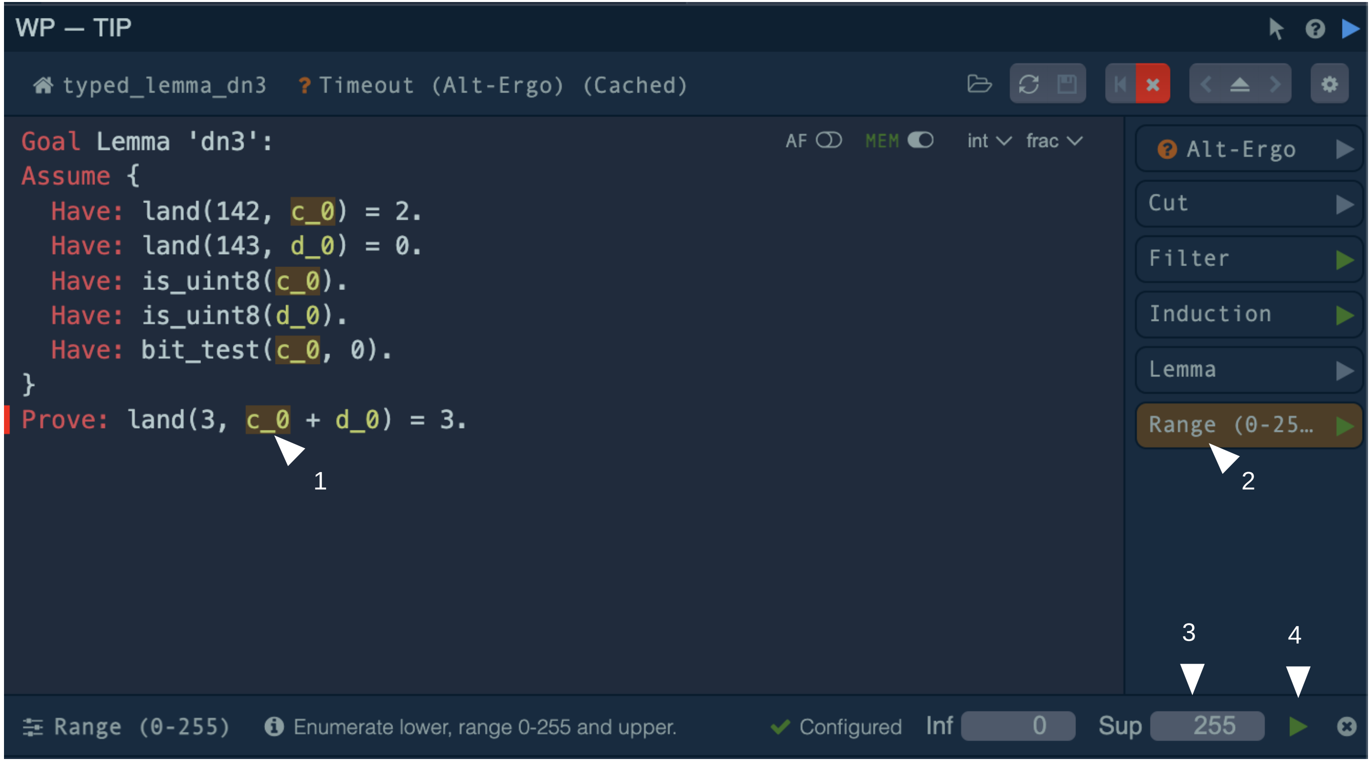
```
lemma dn3:
  ∀ unsigned char c d;
 (c & 0x8E) == 2 ∧
 (c & 0x01) == 1 ∧
 (d & 0x8F) == 0
  ⇒ ((c+d) & 0x03) == 0x03;
```

A proof strategy that generates a script proving the lemma

```
strategy RangeThenProver:
  \tactic ("Wp.range",
    \pattern(is_uint8(e)),
    \select(e),
      \param("inf",0),\param("sup",255),
      \children(RangeThenProver) ),
  \prover("alt-ergo",2);
proof RangeThenProver: dn3;
```


THALES

Demo: Script Tactics



THALES

Demo: Generated script thanks to the provided strategy

```
[ ~/Frama-C/master ]
$ ./bin/frama-c -wp -wp-prover tip ~/work/bits_auto.c
[kernel] Parsing /Users/correnson/work/bits_auto.c (with preprocessing
[wp] 1 goal scheduled
[wp] [Cache] not used
[wp] Proved goals:    1 / 1
  Qed:               0 (6ms)
  Script:            1 (Tactics 9) (Qed 2314/2314 6ms)
[wp] Updated session
   - 1 new valid script
[ ~/Frama-C/master ]
$
```


THALES

New proof strategy mechanism : initial experiments

Applied to the proof of the real-life JCVM code at Thales

- 8,000+ lines of C and 30,000+ lines of ACSL
- Complete proof for 85,000 goals using Alt-Ergo with a 250s timeout requires 800+ proof scripts.

With the new extension: significant time savings

- after a manual creation of strategies (~2 days),
- WP automatically produces more than 50% of the required scripts, whose
- Their manual creation would take ~1 person-month.

An even greater number of proof scripts is expected to be generated from strategies

- This will strongly facilitate industrial verification

THALES

Conclusion

- A new mechanism to automate proof in Frama-C/WP
- Facilitates deductive verification on large projects, avoids time-consuming interactive proof scripts
- Makes the proof more robust w.r.t. changes in the code, spec, tools…
- Promising experimental results on an industrial project at Thales

Future Work

- Extend the strategy language for more complex strategies (e.g. with instantiation)
- A larger evaluation on other projects
- Scaling to large programs having parts with and without low-level operations, or where some of the maintained properties are irrelevant
  - Collaborative memory models
  - More abstract levels of reasoning

References

On proof strategies:

- Loïc Correnson, Allan Blanchard, Adel Djoudi and Nikolai Kosmatov.
  "Automate where Automation Fails: Proof Strategies for Frama-C/WP." **TACAS 2024.** Springer.

On MetAcsl:

- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  "MetAcsl: Specification and Verification of High-Level Properties." **TACAS 2019.** Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  "Tame your annotations with MetAcsl: Specifying, Testing and Proving High-Level Properties". **TAP 2019**. Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
  "Methodology for Specification and Verification of High-Level Properties with MetAcsl". **FormaliSE 2021.** IEEE.

On JavaVard Virtual Machine verification for certification:

- Adel Djoudi, Martin Hana and Nikolai Kosmatov.
  "Formal verification of a JavaCard virtual machine with Frama-C". **FM 2021.** Springer.
- Adel Djoudi, Martin Hána, Nikolai Kosmatov, Milan Kříženecký, Franck Ohayon, Patricia Mouy, Arnaud Fontaine and David Féliot.
  "A Bottom-Up Formal Verification Approach for Common Criteria Certification:
  Application to JavaCard Virtual Machine". **ERTS 2022, Best paper award**.

Back-Up Slides

Common Criteria: Evaluation assurance levels (EAL)

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		1	1	1	1	1	1
	ADV_FSP	1	2	3	4	5	5	6
	ADV_IMP				1	1	2	2
	ADV_INT					2	3	3
	ADV_SPM						1	1
	ADV_TDS		1	2	3	4	5	6
Guidance documents	AGD_OPE	1	1	1	1	1	1	1
	AGD_PRE	1	1	1	1	1	1	1
Life-cycle support	ALC_CMC	1	2	3	4	4	5	5
	ALC_CMS	1	2	3	4	5	5	5
	ALC_DEL		1	1	1	1	1	1
	ALC_DVS			1	1	1	2	2
	ALC_FLR							
	ALC_LCD			1	1	1	1	2
	ALC_TAT				1	2	3	3
Security Target evaluation	ASE_CCL	1	1	1	1	1	1	1
	ASE_ECD	1	1	1	1	1	1	1
	ASE_INT	1	1	1	1	1	1	1
	ASE_OBJ	1	2	2	2	2	2	2
	ASE_REQ	1	2	2	2	2	2	2
	ASE_SPD		1	1	1	1	1	1
	ASE_TSS	1	1	1	1	1	1	1
Tests	ATE_COV		1	2	2	2	3	3
	ATE_DPT			1	1	3	3	4
	ATE_FUN		1	1	1	1	2	2
	ATE_IND	1	2	2	2	2	2	3
Vulnerability assessment	AVA_VAN	1	2	2	3	4	5	5

EAL1	Functionally tested
EAL2	Structurally tested
EAL3	Methodically tested and checked
EAL4	Methodically designed, tested and reviewed
EAL5	Semiformally designed and tested
EAL6	**Semiformally verified design and tested**
EAL7	**Formally verified design and tested**

Source:
CCpart3v3.1 - Table 1
(https://www.commoncriteriaportal.org/cc/)

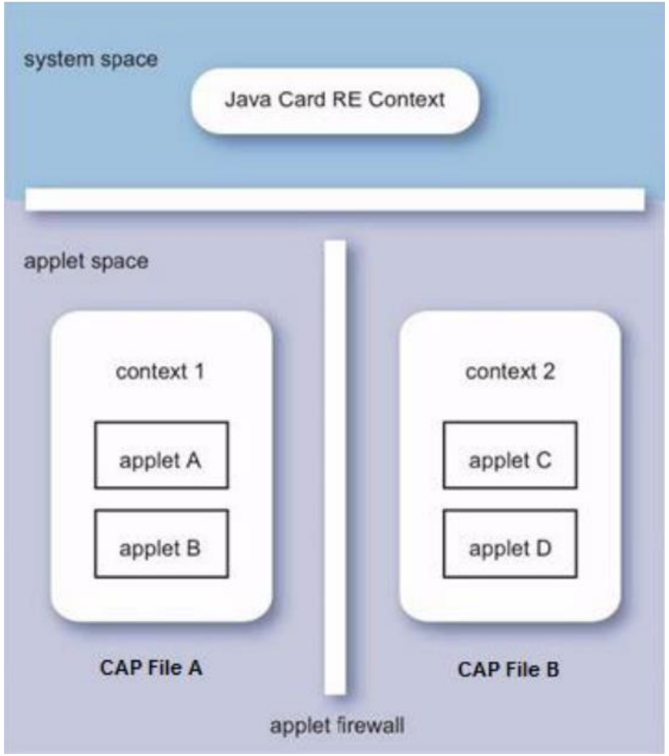
Common Criteria: Certified products (consulted on March 7, 2024)

Certified Products by Assurance Level and Certification Date

EAL	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	Total
Basic	0	0	0	0	0	0	0	0	0	0	1	4	38	44	0	87
EAL1	0	0	0	0	0	0	0	0	0	3	4	3	2	0	1	13
EAL1+	0	0	0	0	0	0	0	0	0	0	1	0	2	2	0	5
EAL2	0	0	0	0	0	0	0	1	0	17	15	39	12	12	1	97
EAL2+	0	0	0	0	0	2	1	5	2	28	43	35	30	33	1	180
EAL3	0	0	0	0	0	2	0	0	0	9	9	4	0	2	2	28
EAL3+	0	0	0	0	0	3	1	0	1	4	12	18	29	13	1	82
EAL4	0	0	0	0	0	0	3	0	5	6	5	3	2	3	0	27
EAL4+	1	0	0	0	1	3	7	5	6	44	60	66	73	90	8	364
EAL5	0	0	0	0	0	0	1	0	0	0	2	0	4	2	0	9
EAL5+	0	0	0	0	2	2	4	17	12	41	69	44	39	77	14	321
EAL6	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	2
EAL6+	0	0	0	0	0	1	0	0	0	20	20	30	33	37	2	143
EAL7	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	3
EAL7+	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2
Medium	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
None	0	0	0	0	0	0	0	0	0	38	44	77	75	113	13	360
US Standard	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Totals:	**1**	**0**	**0**	**0**	**3**	**13**	**17**	**28**	**26**	**210**	**286**	**324**	**341**	**431**	**43**	**1723**

JCVM: Standard Specification (2/2)

- The **Firewall guarantees isolation** of heap data between different contexts
- Java Card Runtime Environment (**JCRE**) context is a **privileged context** devoted to system operations
- **Well-defined exceptions:** global arrays, shareable interfaces,…



EAL6-EAL7: Formal verification of Security Properties

Security Aspect

#.Firewall: "*The Firewall shall ensure controlled sharing of class instances, and* ***isolation of their data and code between packages*** *(that is, controlled execution contexts) as well as between packages and the JCRE context...*"

[Java Card System – Open Configuration Protection Profile – V3.1]

Security properties (simplified examples)

- **integrity_header**: allocated objects' headers cannot be **modified** during a VM run.
- **integrity_data**: allocated objects' data can be **modified** only by the owner.
- **confidentiality_data**: allocated objects' data can be **read** only by the owner.

Evaluation Assurance Levels

EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7

Formal verification (EAL6–EAL7)

Formal verification of security properties

Specification effort

JCVM C code		ACSL Annotations			
		User provided annotations		MetAcsl	RTE
# Functions	# Loc C	# Loc Ghost	# Loc ACSL	# Loc ACSL	# Loc ACSL
381	7,014	162	35,480	396,603	2,290

Large code

A few yet necessary

12,432 before preprocessing macros that gather redundant annotations
Still a considerable effort

Automatically generated from **36** metaproperties only

- **User-provided annotations**: predicates, lemmas, function contracts, loop contracts and other assertions
- **MetAcsl**: automatically generated annotations according to user-defined metaproperties
- **RTE:** automatically generated annotations in order to prevent undefined behaviors

THALES

Some Issues (I), Solutions (S) and Perspectives (P)

Companion ghost model

- **I**: **Automatic proof fails** on low-level code (bit-fields)
- **S:** Linking bits to ghost integer variables brings the **prover back into its comfort zone**
- **P: Proof at the abstract level** for some properties can help [as discussed at Dagstuhl]

Proof scripts for complex predicates

- **I: Automatic proof fails** to use the right predicates
- **S: Guide the first proof steps** by unfolding relevant predicates or instantiating values
- **S&P: New proof strategy mechanism** to generate scripts automatically [TACAS'24]

Carefully chosen lemmas

- **I: Automatic proof fails** repeatedly in similar cases
- **S:** Lemmas help to **re-use the same reasoning**

THALES