Advances in Programming Languages and in Program Verification

Habilitation Thesis

Ștefan Ciobâcă

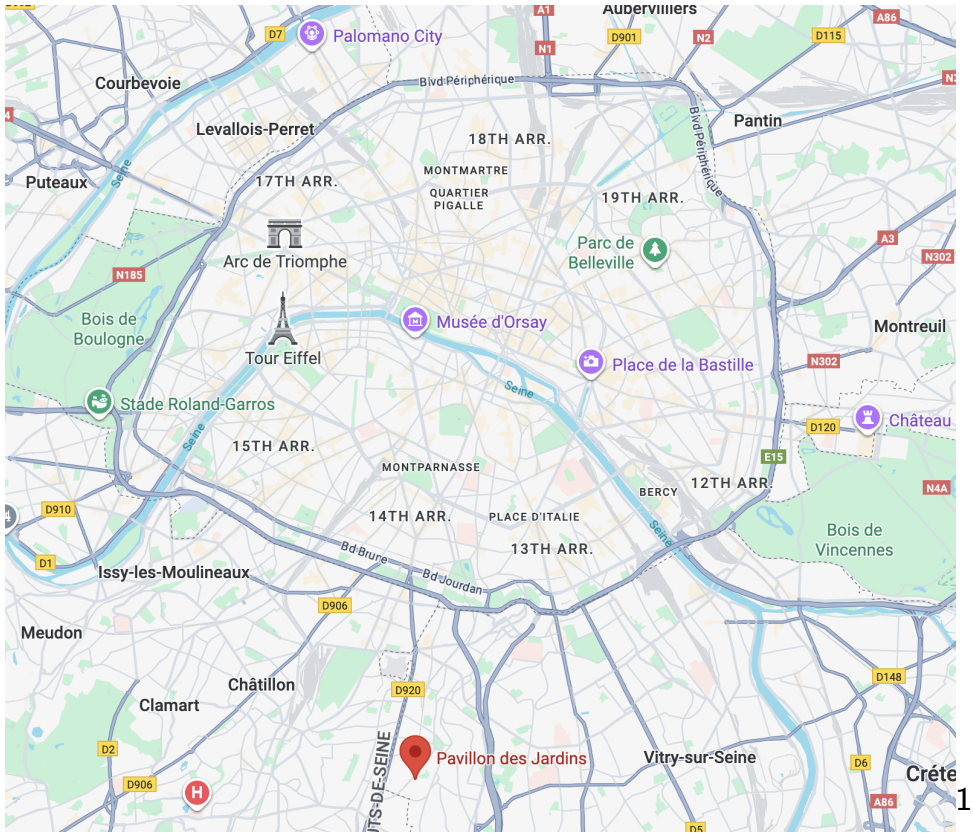
Alexandru Ioan Cuza University, Iași, Romania

September 19th, 2025

Outline

PhD Studies

Laboratoire Spécification et Vérification, ENS Cachan 2008-2011



PhD Thesis: *Verification and Composition of Security Protocols with Applications to Electronic Voting*

Advisors: Véronique Cortier, Steve Kremer, and Jean Goubault-Larrecq.

[1] Maps source: Google Maps

Verification of Security Protocols

$$\underset{\text{e.g., TLS, Kerberos, OAuth}}{\textit{Protocol } P} \models \underset{\text{e.g., does not leak a key } k}{\textit{Property } \varphi}$$

- Protocol (symbolic model):

$$\begin{array}{lllll}
 & out(A, s(b(c(u, r_A), b_A), k_A)). & & & \\
A(u) & in(A, x_A). & out(c, u(x_A, b_A)). & out(c, r_A) & \\
 & check(x_A, pk(k)) = b(c(u, r_A), b_A) & & & \\
 & \quad \| & \quad . \quad \| & \quad . \quad \| & \\
 & out(B, s(b(c(v, r_B), b_B), k_B)). & & & \\
B(v) & in(B, x_B). & out(c, u(x_A, b_A)). & out(c, r_B) & \\
 & check(x_B, pk(k)) = b(c(v, r_B), b_B) & & &
\end{array}$$

- Primitives (equational theory):

$$\begin{aligned}
&o(c(x, y), y) = x \\
&check(s(x, y), pk(y)) = x \\
&\ldots
\end{aligned}$$

- Property (behavioral equivalence):

$$A(u) \parallel B(v) \equiv A(v) \parallel B(u)$$

Verification of Security Protocols

• **CADE 2009**: Ștefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. *Computing knowledge in security protocols under convergent equational theories* (extended version in **JAR 2012**)

Input: $X = x_1, x_2, \ldots, x_n$, $Y = y_1, y_2, \ldots, y_n$ Output: $X \stackrel{?}{\equiv} Y$

• **ESOP 2012**: Rohit Chadha, Ștefan Ciobâcă, and Steve Kremer. *Automated verification of equivalence properties of cryptographic protocols*

Input: A, B *(processes)* Output: $A \stackrel{?}{\equiv} B$

• **CSF 2010**: Ștefan Ciobâcă and Véronique Cortier. *Protocol composition for arbitrary primitives.*

$$\begin{array}{l} P \models \varphi \\ Q \models \varphi \end{array} \implies P \parallel Q \stackrel{?}{\models} \varphi$$

Verification of Security Protocols

ESOP 2012: Rohit Chadha, Ștefan Ciobâcă, and Steve Kremer. *Automated verification of equivalence properties of cryptographic protocols*

• Key idea 1 (encoding protocols as Horn clauses): $in(x).out(enc(x, k))$

$$w_1 \rhd_{in(x).out} enc(x,\ k) \mid X \rhd_\epsilon x.$$

• Key idea 2 (encoding rewriting theory as Horn clauses): *variants* of terms: $o(x, z)$ has a variant of z for the case where $x = c(z, y)$

$$o(X, Y) \rhd z \mid X \rhd c(z, y), Y \rhd y$$

UNIF 2011 Ștefan Ciobâcă.
Computing finite variants for subterm convergent rewrite systems

• Key idea 3 (solving using theorem proving): tailored prover based on a refinement of first-order resolution

`https://github.com/ciobaca/akiss`

Verification of Security Protocols (Post PhD)

Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23, 2016 (extended version of **ESOP 2012**)

• 2014: Cholewa, Meseguer, and Escobar. *Variants of Variants and the Finite Variant Property*

"Ciobâcă Variants (Ciob-Variants)"

• Software

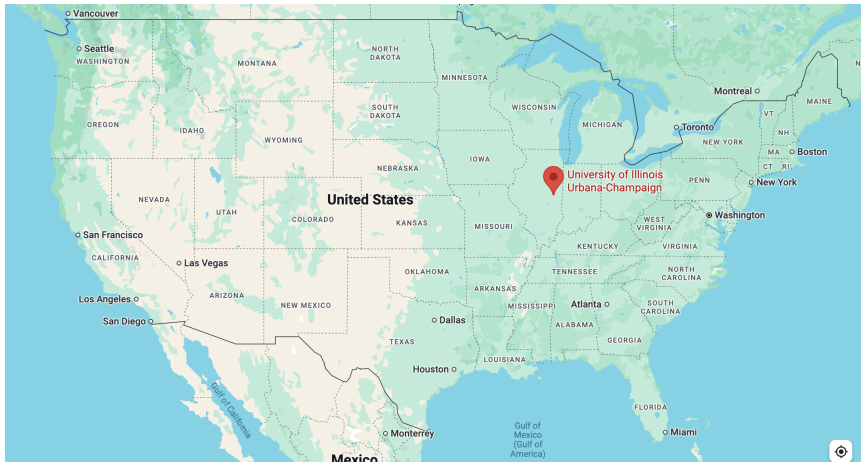
`https://github.com/ciobaca/kiss`
`https://profs.info.uaic.ro/stefan.ciobaca/subvariant/`
`https://github.com/ciobaca/akiss`

March 2025: King's College in London: possibility of using (A-)KiSS for "Dolev Yao as a service (or as a plug-in)".

Outline

2011: DAK Project
`kframework.org`

UIUC



UAIC

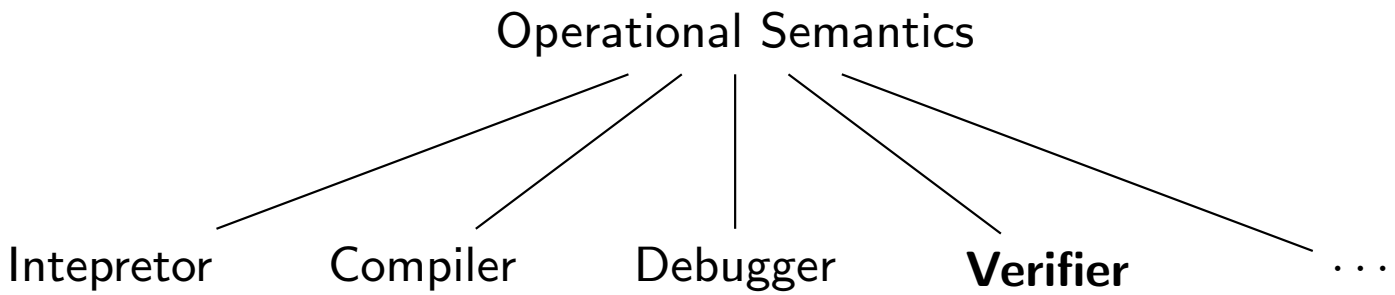
K Framework: Programming Language Semantics

```
syntax AExp ::= Id | Int
              | "(" AExp ")"       [bracket]
              | AExp "+" AExp      [left, strict]
syntax Stmt ::= "{" "}"
              | "{" Stmt "}"
              | Id "=" AExp ";"    [strict(2)]
              | "if" BExp
                "then" Stmt
                "else" Stmt        [strict(1)]
              | "while" BExp "do" Stmt
              > Stmt Stmt          [left]

configuration <T color="yellow">
                <k color="green"> $PGM:Pgm </k>
                <state color="red"> .Map </state>
              </T>

rule <k> X = I:Int; => . ...</k>
     <state>... X |-> (_ => I) ...</state>
```

K Framework: Idea



Matching Logic

- Logic to reason about programs initially developed by Roșu and Schulte.

Matching Logic Formulae: $x \wedge x > 3$;

Satisfaction Relation: $\gamma, \rho \models \varphi$;

Example: $4, \{x \mapsto 4\} \models x \wedge x > 3$.

Reachability Logic

A reachability logic formula:

$$\varphi \Rightarrow \varphi'$$

- operational semantics rule:

```
<k> X = I:Int; => . ...</k>
<state>... X |-> (_ => I) ...</state>
```

- program specification:

```
<k> SUM </k> <state> n |-> n </state>
=>
<k> .  </k> <state> n |-> n s |-> n(n + 1) ÷ 2 ...  </state>
```

Reachability Logic

$$\textsc{Circularity}\ \frac{A \vdash_{C \cup \{\varphi \Rightarrow \varphi'\}} \varphi \Rightarrow \varphi'}{A \vdash_C \varphi \Rightarrow \varphi'}$$

$$\textsc{Transitivity}\ \frac{A \vdash_C \varphi \Rightarrow^+ \varphi' \qquad A \cup C \vdash \varphi' \Rightarrow \varphi''}{A \vdash_C \varphi \Rightarrow \varphi''}$$

Grigore Roșu and Andrei Ștefănescu. Checking reachability using matching logic. In Gary T. Leavens and Matthew B. Dwyer, editors, *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012*, pages 555–574. ACM, 2012

Reachability Logic

Typical induction proof does not work:

$$A \vdash_C \varphi \Rightarrow \varphi' \rightarrow A \models_C \varphi \Rightarrow \varphi'$$

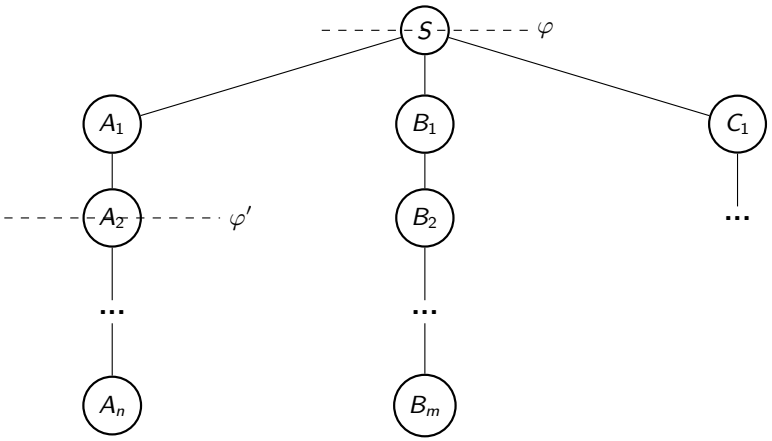
$$\begin{array}{l} A \vdash_C \varphi \Rightarrow \varphi' \rightarrow \\ \quad \forall \gamma^0. \\ \quad\quad S \models_{\gamma^0} A \wedge S \models^+_{\gamma^0} C \rightarrow \\ \quad\quad\quad \forall \gamma, \rho. \\ \quad\quad\quad\quad \gamma^0 \Rightarrow \gamma \wedge \gamma, \rho \models \varphi \rightarrow \\ \quad\quad\quad\quad\quad \exists \gamma'.(\gamma \Rightarrow \gamma' \wedge \gamma', \rho \models \varphi'), \end{array}$$

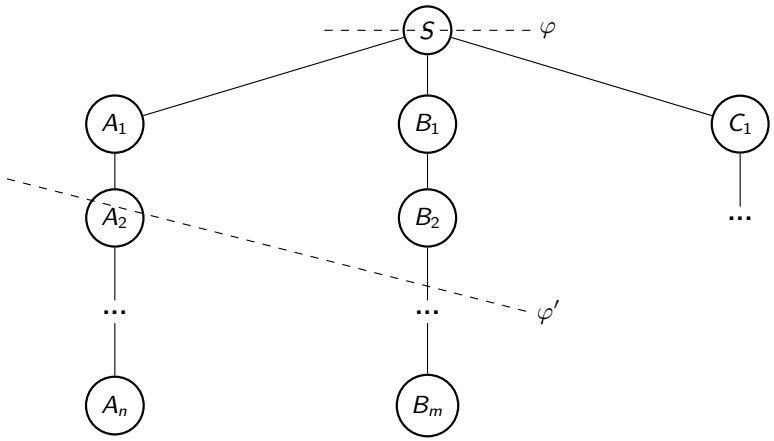
Reachability Logic (One-Path)

```
Theorem soundness :
  WeaklyWDSystem S ->
  forall A C phi phi',
    PS A C phi phi' ->
    forall g,
      Terminates S g ->
      GStronglyValidssys true A g ->
      GAlmostStronglyValidssys true C g ->
      ((IsEmpty C ->
        GStronglyValid false (phi, phi') g) /\
      ((not (IsEmpty C)) ->
        GStronglyValid true (phi, phi') g)).
```

Grigore Roșu, Andrei Ștefănescu, Ștefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 358–367. IEEE Computer Society, 2013

Reachability Logic (All-Path)





$$\textsc{Step}\ \frac{\models \varphi \rightarrow \bigvee_{\varphi_l \Rightarrow^{\exists} \varphi_r \in S} \exists \mathit{FreeVars}(\varphi_l).\varphi_l \qquad \models \exists c.(\varphi[c/\Box] \wedge \varphi_l[c/\Box]) \wedge \varphi_r \rightarrow \varphi'}{S, A \vdash_C \varphi \Rightarrow^{\forall} \varphi'}$$

Andrei Ștefănescu, Ștefan Ciobâcă, Radu Mereuță, Brandon M. Moore, Traian-Florin Șerbănuță, and Grigore Roșu. All-path reachability logic. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2014 (extended version in **LMCS 2019**)

Transforming Semantics

Small-step style:

$$\frac{e_1 \longrightarrow e_1'}{e_1 + e_2 \longrightarrow e_1' + e_2} \qquad \frac{e_2 \longrightarrow e_2'}{v_1 + e_2 \longrightarrow v_1 + e_2'} \qquad \frac{}{v_1 + v_2 \longrightarrow v_1 +_{Int} v_2}$$

Big-step style:

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 + e_2 \Downarrow v_1 +_{Int} v_2}$$

Transforming Semantics

- Transform

$$\frac{M_1 \longrightarrow N_1 \qquad \ldots \qquad M_n \longrightarrow N_n}{M \longrightarrow N}\ \phi$$

into

$$\frac{M_1 \Downarrow N_1 \qquad \ldots \qquad M_n \Downarrow N_n \qquad N \Downarrow V}{M \Downarrow V}\ \phi,$$

- and add

$$\frac{}{V \Downarrow V.}\ V\downarrow$$

Conditions:

- ground confluence for the small-step relation;
- values (defined by $\downarrow$) are normal forms w.r.t $\longrightarrow$;
- star-soundness;
- star-completeness.

Ștefan Ciobâcă. From small-step semantics to big-step semantics, automatically. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2013

Impact

- Reachability Logic:

  Runtime Verification, Inc.: `https://runtimeverification.com`

- **iFM 2013**

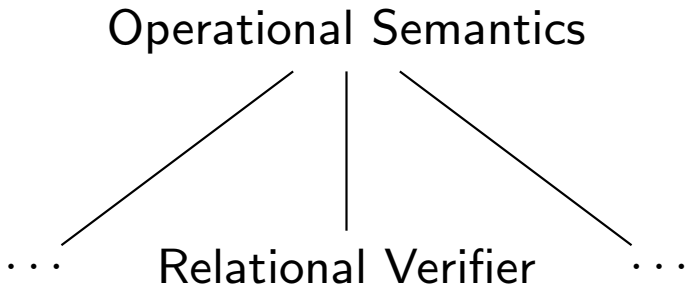
  talk at **SSLF 2012** (citations in ESOP 2014, . . ., ICFP 2025)

  2014: Seminar Programmiersprachen (Freie Universität Berlin, Prof. Dr. E. Fehr, Lilit Hakobyan)

  TYPES 2017 Ștefan Ciobâcă and Vlad Andrei Tudose. *Automatically constructing a type system from the small-step semantics*

Outline

Proofs of Relational Properties



Relational properties = Properties of more than one program

Program equivalence = For the same input, the two programs produce the same output

Reducing Equivalence To Correctness

Start with two programming languages:
- $Cfg_L, S_L, \Sigma_L, \mathcal{T}_L, \mathcal{A}_L$;
- $Cfg_R, S_R, \Sigma_R, \mathcal{T}_R, \mathcal{A}_R$.

Construct the product language: $Cfg, S, \Sigma, \mathcal{T}, \mathcal{A}$.

Lemma (**Lemma 1** in the SYNASC 2014 paper)

We have that

$$(\gamma_L, \gamma_R) \rightarrow_{\mathcal{A}} (\gamma'_L, \gamma'_R)$$

if either

$$\gamma_L \rightarrow_{\mathcal{A}_L} \gamma'_L$$

or

$$\gamma_R \rightarrow_{\mathcal{A}_R} \gamma'_R.$$

Reducing Equivalence To Correctness

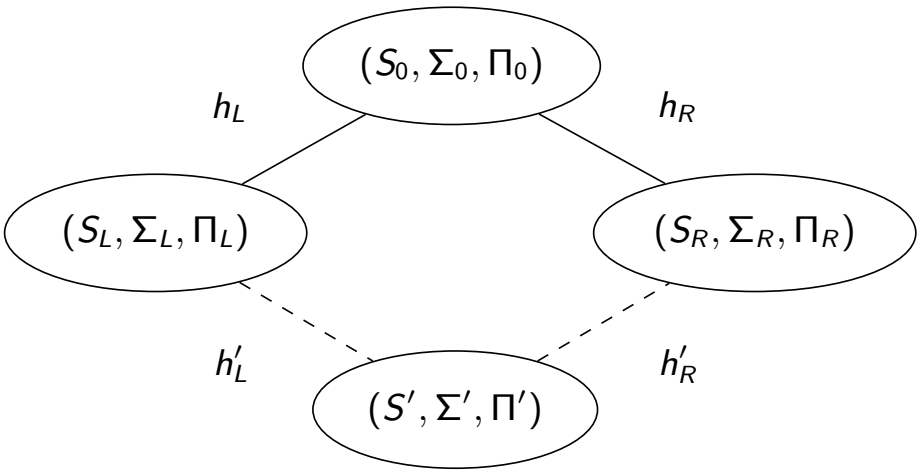
Theorem (Theorem 4 in the SYNASC 2014 paper)

Two programs P_L and P_R are partially equivalent iff

$$\mathcal{A} \models (input_L(i), input_R(i)) \Rightarrow (C_L, C_R) \wedge output_L(C_L) = output_R(C_R).$$

Ștefan Ciobâcă. Reducing partial equivalence to partial correctness. In Franz Winkler, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, Timisoara, Romania, September 22-25, 2014*, pages 164–171. IEEE Computer Society, 2014

Programming Languages Aggregation



Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A theoretical foundation for programming languages aggregation. In Mihai Codescu, Razvan Diaconescu, and Ionut Tutu, editors, *Recent Trends in Algebraic Development Techniques - 22nd International Workshop, WADT 2014, Sinaia, Romania, September 4-7, 2014, Revised Selected Papers*, volume 9463 of *Lecture Notes in Computer Science*, pages 30–47. Springer, 2014

$$A_a = \{\iota_y^L(\varphi_1) \Rightarrow \iota_y^L(\varphi_1') \mid \varphi_1 \Rightarrow \varphi_1' \in A_L\} \cup \{\iota_x^R(\varphi_2) \Rightarrow \iota_x^R(\varphi_2') \mid \varphi_2 \Rightarrow \varphi_2' \in A_R\}$$

$$A_p = \{\iota_y^L(\varphi_1) \wedge \iota_x^R(\varphi_2) \Rightarrow \exists x.\exists y.(\iota_y^L(\varphi_1') \wedge \iota_x^R(\varphi_2')) \mid \varphi_1 \Rightarrow \varphi_1' \in A_L, \varphi_2 \Rightarrow \varphi_2' \in A_R\}$$

$$A = A_a \cup A_p$$

Full Equivalence

```
c := n;
n := 1;
while (c != 1)
  n := n + 1;
  if (c % 2 != 0)
    then c := 3 * c + 1
    else c := c / 2
```

μf.λn.λa.

```
  if n != 1
    then
      if n % 2 != 0
        then f (3 * n + 1) (a + 1)
        else f (n / 2) (a + 1)
      else
        a
```

Full Equivalence

$$\textsc{Axiom}\ \frac{\varphi \in E}{\vdash \varphi \Downarrow^{\infty} E} \qquad \textsc{Step}\ \frac{\models \varphi_1 \Rightarrow^{*} \varphi_1' \qquad \models \varphi_2 \Rightarrow^{*} \varphi_2' \qquad \vdash \langle \varphi_1', \varphi_2' \rangle \Downarrow^{\infty} E}{\vdash \langle \varphi_1, \varphi_2 \rangle \Downarrow^{\infty} E}$$

$$\textsc{Conseq}\ \frac{\varphi \rightarrow \exists \tilde{x}.\varphi' \qquad \varphi' \Downarrow^{\infty} E}{\vdash \varphi \Downarrow^{\infty} E} \qquad \textsc{Case Analysis}\ \frac{\varphi \Downarrow^{\infty} E \qquad \varphi' \Downarrow^{\infty} E}{\vdash \varphi \vee \varphi' \Downarrow^{\infty} E}$$

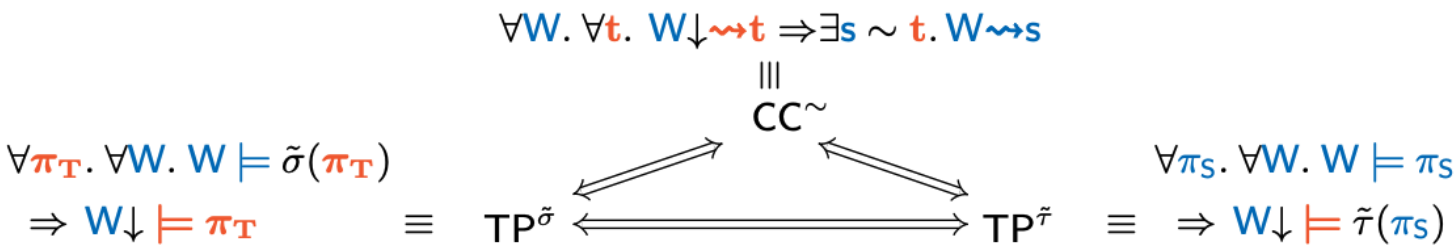
$$\textsc{Circularity}\ \frac{\models \varphi_1 \Rightarrow^{+} \varphi_1' \qquad \models \varphi_2 \Rightarrow^{+} \varphi_2' \qquad \vdash \langle \varphi_1', \varphi_2' \rangle \Downarrow^{\infty} E \cup \{\langle \varphi_1, \varphi_2 \rangle\}}{\vdash \langle \varphi_1, \varphi_2 \rangle \Downarrow^{\infty} E}$$

Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A language-independent proof system for mutual program equivalence. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2014

Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A language-independent proof system for full program equivalence. *Formal Aspects Comput.*, 28(3):469–497, 2016

Trace-Relating Compiler Correctness and Secure Compilation

- compiler correctness = any **trace** of the target program **is** also a **trace** of the source program
- CompCert: additional undefined behavior event in the source
- CakeML: additional resource exhaustion event in the target

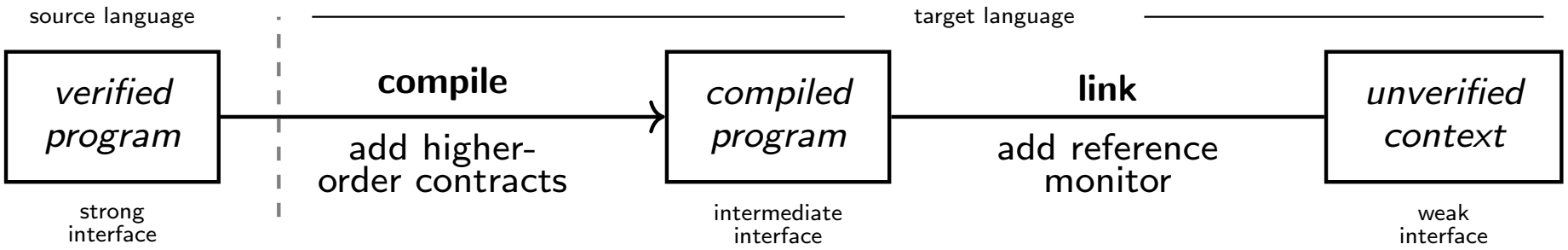


- secure compilation = if source program has a security property, then target program also has the same property

the trinitarian view for compiler correctness extends to a number of definitions for *secure compilation*

Carmine Abate, Roberto Blanco, Ștefan Ciobâcă, Adrien Durier, Deepak Garg, Cătălin Hriţcu, Marco Patrignani, Éric Tanter, and Jérémy Thibault. An extended account of trace-relating compiler correctness and secure compilation. *ACM Trans. Program. Lang. Syst.*, 43(4):14:1–14:48, 2021 (extended version of ESOP 2020 paper)

Secure IO (SCIO*)



Cezar-Constantin Andrici, Ștefan Ciobâcă, Cătălin Hriţcu, Guido Martínez, Exequiel Rivas, Éric Tanter, and Théo Winterhalter. Securing verified IO programs against unverified code in F*. *Proc. ACM Program. Lang.*, 8(POPL):2226–2259, 2024

Impact

- language aggregation construction shows the power of language-parametric tools;
- Dagstuhl Seminar 18151 on Program Equivalence;
- PERR (Program Equivalence and Relational Reasoning);
- starting point for collaboration with Andrei-Sebastian Buruiană;
- Cezar-Constantin Andrici pursued a PhD thesis with Cătălin Hriţcu.

Outline

The IZA Project

- Bridge Grant between between the Alexandru Ioan Cuza University and Bitdefender.
- Goal: transfer the expertise of the FMSE (Formal Methods in Software Engineering) group in verification and static analysis to Bitdefender.

Which static analyzer should we use?

- An example of a test case in the Toyota ITC [23] test suite:

```
void bit_shift_001 ()
{
  int a = 1;
  int ret;
  ret = a << 32;
  /*ERROR:
    Bit shift error*/
  sink = ret;
}
```

```
void bit_shift_001 ()
{
  int a = 1;
  int ret;
  ret = a << 10;
  /*NO ERROR:
    Bit shift error*/
  sink = ret;
}
```

The IZA Project

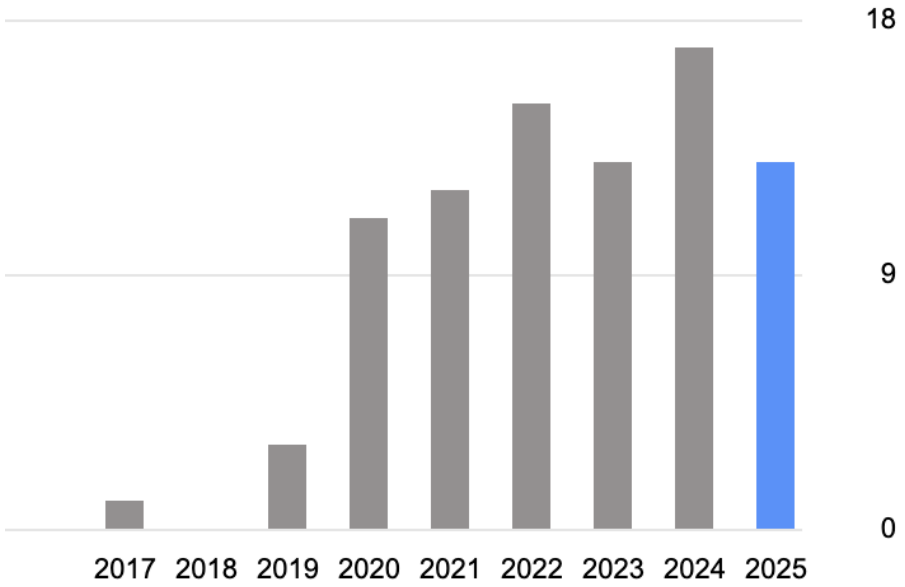
Tool	D1	D2	D3	D4	D5	D6	D7	D8	D9
System	0	0	5	0	0	0	0	0	1
Clang	15	1	9	1	9	8	2	1	13
Cppcheck	0	0	0	0	0	0	1	0	0
Flawfinder	0	0	0	0	0	0	0	1	0
Flint++	0	0	0	0	0	0	0	0	0
Frama-C	0	28	0	6	8	9	1	0	0
Infer	0	0	0	0	0	1	3	0	0
Oclint	0	0	0	0	0	0	0	0	0
Sparse	0	0	0	0	0	0	0	0	0
Splint	0	0	2	0	3	1	1	0	0
Uno	0	0	0	0	0	1	0	0	2

• D1 = Concurrency defects, D2 = Dynamic memory defects, D3 = Inappropriate code, D4 = Misc defects, D5 = Numerical defects, D6 = Pointer related defects, D7 = Resource management defects, D8 = Stack related defects, D9 = Static memory defects

Andrei Arusoaie, Ștefan Ciobâcă, Vlad Craciun, Dragos Gavrilut, and Dorel Lucanu. A comparison of open-source static analysis tools for vulnerability detection in C/C++ code. In Tudor Jebelean, Viorel Negru, Dana Petcu, Daniela Zaharie, Tetsuo Ida, and Stephen M. Watt, editors, *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017*, pages 161–168. IEEE Computer Society, 2017

Impact

- Citations[2]:



- March 2025:

Contact from Lund University

[2]Source: Google Scholar

Outline

Logically Constrained Term Rewriting Systems

K framework: most results do not make use of matching logic in its full generality.

Is there a simpler, but still useful, formalism?

A logically constrained term rewriting rule [18] is of the form

$$l \longrightarrow r \text{ if } \phi,$$

where l and r are terms and ϕ is a (typically first-order) constraint.

LCTRSs can be used to define operational semantics.

Reducing Total Correctness to Partial Correctness

$$\underbrace{V_t}_{\text{total correctness}}(S, P) = \underbrace{V}_{\text{partial correctness}}(\Theta(S), \Theta(P))$$

Main idea: for every rule $\varphi \Rightarrow \varphi'$ add a rank: $\Theta(\varphi, n) \Rightarrow \Theta(\varphi, n-1)$.

Theorem (Theorem 3.1 (Page 9) in our WPTE paper)

If there exists some term $s \in Term_{\Sigma,Nat}(Var)$ *of sort Nat such that*

$$\Theta(S) \models \Theta(\varphi, s) \Rightarrow^{\forall} \exists M.\Theta(\varphi', M),$$

where $M \in Var_{Nat}$, *then*

$$S \models_t \varphi \Rightarrow^{\forall} \varphi'.$$

Andrei-Sebastian Buruiană and Ștefan Ciobâcă. Reducing total correctness to partial correctness by a transformation of the language semantics. In Joachim Niehren and David Sabel, editors, *Proceedings Fifth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2018, Oxford, England, 8th July 2018*, volume 289 of *EPTCS*, pages 1–16, 2018

A coinductive approach to reachability in LCTRSs

$$\langle t \mid \phi \rangle \Rightarrow \langle t' \mid \phi' \rangle$$

$$\text{[axiom]}\ \frac{}{\langle\ t_l\ \mid\ \bot\ \rangle \Rightarrow \langle\ t_r\ \mid\ \phi_r\ \rangle}$$

$$\text{[subs]}\ \frac{\langle\ t_l\ \mid\ \phi_l \wedge \neg(\exists \tilde{x}.t_l = t_r \wedge \phi_r)\ \rangle \Rightarrow \langle\ t_r\ \mid\ \phi_r\ \rangle}{\langle\ t_l\ \mid\ \phi_l\ \rangle \Rightarrow \langle\ t_r\ \mid\ \phi_r\ \rangle}$$

$$\textit{where}\ \begin{array}{l} \widetilde{x} \triangleq \mathit{var}(t_r, \phi_r) \setminus \mathit{var}(t_l, \phi_l) \\ \exists \tilde{x}.t_l = t_r \wedge \phi_r \text{ satisfiable} \end{array}$$

$$\text{[der}^{\forall}\text{]}\ \frac{\langle\ t^j\ \mid\ \phi^j\ \rangle \Rightarrow \langle\ t_r\ \mid\ \phi_r\ \rangle, j \in \{1, \ldots, n\}}{\langle\ t_l\ \mid\ \phi_l\ \rangle \Rightarrow \langle\ t_r\ \mid\ \phi_r\ \rangle}$$

$$\textit{where}\ \begin{array}{l} \langle\ t_l\ \mid\ \phi_l\ \rangle \text{ is } \mathcal{R}\text{−derivable and} \\ \phi_l \rightarrow \bigvee_{j \in \{1,\ldots,n\}} \exists \widetilde{y}^j.\phi^j \text{ is valid} \\ \Delta_{\mathcal{R}}(\langle\ t_l\ \mid\ \phi_l\ \rangle) = \{\langle\ t^1\ \mid\ \phi^1\ \rangle, \ldots, \langle\ t^n\ \mid\ \phi^n\ \rangle\} \text{ and} \\ \widetilde{y}^j = \mathit{var}(t^j, \phi^j) \setminus \mathit{var}(t_l, \phi_l) \end{array}$$

Theorem

The proof system is sound and complete.

A coinductive approach to reachability in LCTRSs

$$\text{[circ]}\ \frac{\langle\ t_r^c\ \mid\ \phi_l \wedge \phi \wedge \phi_r^c\ \rangle \Rightarrow \varphi_r, \quad \langle\ t_l\ \mid\ \phi_l \wedge \neg\phi\ \rangle \Rightarrow \varphi_r}{\langle\ t_l\ \mid\ \phi_l\ \rangle \Rightarrow \varphi_r} \qquad \begin{array}{l}\phi \text{ is } \exists var(t_l^c, \phi_l^c).\, t_l = t_l^c \wedge \phi_l^c, \\ \langle\ t_l^c\ \mid\ \phi_l^c\ \rangle \Rightarrow \langle\ t_r^c\ \mid\ \phi_r^c\ \rangle \in G\end{array}$$

Theorem

If all of the reachability formulae in the set G are provable using guarded proof trees (i.e., trees where CIRC *is used only after* $\text{DER}^{\forall}$*), then all reachability formulae in G are also valid.*

Ștefan Ciobâcă and Dorel Lucanu. A coinductive approach to proving reachability properties in logically constrained term rewriting systems. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2018

Unification Modulo Builtins

$$\mathsf{n} \mapsto 2 \times N + 1, \mathsf{cnt} \mapsto C \quad = \quad \mathsf{cnt} \mapsto C' + N', \mathsf{n} \mapsto N' + 3$$

1: **function** UNIFICATION(t_1, t_2)
2: ▷ returns: a complete set of E-unifiers modulo builtins of t_1 and t_2
3: compute $\langle\ s_1 \mid \phi^{\sigma_1}\ \rangle$, an abstraction of t_1
4: compute $\langle\ s_2 \mid \phi^{\sigma_2}\ \rangle$, an abstraction of t_2
5: compute $\{\tau_1, \ldots, \tau_n\}$, a complete set of E-unifiers of s_1 and s_2
6: **for** $i \in \{1, \ldots, n\}$ **do**
7: $\tau_i' \leftarrow \tau_i|_{\mathcal{X} \setminus \mathcal{X}^{\mathsf{b}}}$
8: $\phi_i' \leftarrow \phi^{\sigma_1} \wedge \phi^{\sigma_2} \wedge \bigwedge_{x \in dom(\tau_i) \cap \mathcal{X}^{\mathsf{b}}} \tau_i(x) = x$
9: **return** $\{(\tau_1', \phi_1'), \ldots, (\tau_n', \phi_n')\}$

Ștefan Ciobâcă, Andrei Arusoaie, and Dorel Lucanu. Unification modulo builtins. In Lawrence S. Moss, Ruy J. G. B. de Queiroz, and Maricarmen Martínez, editors, *Logic, Language, Information, and Computation - 25th International Workshop, WoLLIC 2018, Bogota, Colombia, July 24-27, 2018, Proceedings*, volume 10944 of *Lecture Notes in Computer Science*, pages 179–195. Springer, 2018

Revisiting AC-unification

$$f(x, f(y, z)) = f(f(x, y), z) \qquad f(x, y) = f(y, x)$$

$$\textit{Topmost AC-unification: } f^*(u_1, \ldots, u_l) = f^*(v_1, \ldots, v_k)$$

$$\textit{LDE: } a_1x_1 + \ldots + a_nx_n = b_1x_1 + \ldots + b_nx_n$$

• benchmark four algorithms for solving LDEs (a lexicographic enumeration algorithm, a completion procedure, a graph-based algorithm, and the Slopes algorithm);
• implement AC-unfication as a library.

Valeriu Motroi and Ștefan Ciobâcă. A note on the performance of algorithms for solving linear Diophantine equations in the naturals. *CoRR*, abs/2104.05200, 2021

Valeriu Motroi and Ștefan Ciobâcă. A typo in the Paterson-Wegman-de Champeaux algorithm. *CoRR*, abs/2007.00304, 2020

Operationally-based program equivalence proofs using LCTRSs

Motivating example:

- $f = \lambda n.\ \texttt{if}\ n = 0\ \texttt{then}\ 0\ \texttt{else}\ n + f(n-1)$;
- $F = \lambda n.\lambda i.\lambda a.\ \texttt{if}\ i \leq n\ \texttt{then}\ F(n, i+1, a+i)\ \texttt{else}\ a$.

```
f(3) =
f(2), 3+□ =
f(1), 2+□, 3+□ =
f(0), 1+□, 2+□, 3+□ =
0, 1+□, 2+□, 3+□ =
1, 2+□, 3+□ =
3, 3+□ =
6
```

```
F(3, 0, 0) =
F(3, 1, 0 + 0) =
F(3, 2, 0 + 0 + 1) =
F(3, 3, 0 + 0 + 1 + 2) =
F(3, 4, 0 + 0 + 1 + 2 + 3) =
6
```

Operationally-based program equivalence proofs using LCTRSs

$$\textsc{Axiom}\ \frac{}{G, B \vdash^{g} P \precdot\!\!\sim Q \ \mathit{if}\ \bot}$$

$$\textsc{Base}\ \frac{G, B \vdash^{g} P \precdot\!\!\sim Q \ \mathit{if}\ \phi \wedge \neg\phi_B}{G, B \vdash^{g} P \precdot\!\!\sim Q \ \mathit{if}\ \phi}$$

$$\mathit{if}\ \models \phi_B \rightarrow \bigvee_{Q' \ \mathit{if}\ \phi' \in \Delta^{\leq K}_{\overline{\mathcal{R}}_R}(Q)} \phi' \rightarrow \mathit{sub}\Big((P, Q'), B\Big)$$

$$\textsc{Circ}^{\prec}\ \frac{G, B \vdash^{1} P \prec Q \ \mathit{if}\ \phi \wedge \neg\phi_G}{G, B \vdash^{1} P \prec Q \ \mathit{if}\ \phi}$$

$$\mathit{if}\ \models \phi_G \rightarrow \bigvee_{Q' \ \mathit{if}\ \phi' \in \Delta^{\leq K}_{\overline{\mathcal{R}}_R}(Q)} \phi' \rightarrow \mathit{sub}\Big((P, Q'), G\Big)$$

$$\textsc{Circ}^{\preceq}\ \frac{G, B \vdash^{g} P \preceq Q \ \mathit{if}\ \phi \wedge \neg\phi_G}{G, B \vdash^{g} P \preceq Q \ \mathit{if}\ \phi}$$

$$\mathit{if}\ \models \phi_G \rightarrow \bigvee_{Q' \ \mathit{if}\ \phi' \in \Delta^{\geq 1-g, \leq K}_{\overline{\mathcal{R}}_R}(Q)} \phi' \rightarrow \mathit{sub}\Big((P, Q'), G\Big)$$

$$\textsc{Step}\ \frac{\begin{array}{c} G, B \vdash^{1} P^i \precdot\!\!\sim Q \ \mathit{if}\ \phi^i \ (\text{for all } 1 \leq i \leq n) \\ G, B \vdash^{g} P \precdot\!\!\sim Q \ \mathit{if}\ \phi \wedge \neg\phi^1 \wedge \ldots \wedge \neg\phi^n \end{array}}{G, B \vdash^{g} P \precdot\!\!\sim Q \ \mathit{if}\ \phi}$$

$$\mathit{if}\ \ \Delta_{\mathcal{R}_L}(P \ \mathit{if}\ \phi) = \{P^i \ \mathit{if}\ \phi^i \mid 1 \leq i \leq n\}$$

Operationally-based program equivalence proofs using LCTRSs

- Motivating example: $f \prec F, f \preceq F, F \preceq f$, but cannot show $F \prec f$
- Bounded stack: $\langle x := e; es, env, fs\rangle \longrightarrow \langle e; x := \square; es, env, fs\rangle$ *if* $\neg val(e) \wedge \underbrace{len(es) < k}_{\textit{constrain stack size}}$

  f no longer equivalent to F
- Optimization proofs using programs schemas:

Optimization	PEC	CORK	RMT
Code hoisting	✓	0.32s	0.41s
Constant propagation	✓	0.33s	0.31s
Copy propagation	✓	0.33s	0.26s
If-conversion	✓	0.34s	0.48s
Partial redundancy elimination	✓	0.34s	0.75s
Loop invariant code motion	✓	3.48s	3.79s
Loop peeling	✓	3.26s	0.97s
Loop unrolling	✓	12.17s	7.09s
Loop unswitching	✓	8.19s	4.71s
Software pipelining	✓	8.02s	3.56s
Loop fission	$\checkmark_p$	23.45s	○ 10.40s
Loop fusion	$\checkmark_p$	23.34s	○ 9.67s
Loop interchange	$\checkmark_p$	29.30s	□ 108.63s
Loop reversal	$\checkmark_p$	8.41s	2.70s
Loop skewing	$\checkmark_p$	8.50s	7.68s
Loop flattening	×	×	□ 8.14s
Loop strength reduction	×	5.63s	5.26s
Loop tiling 01	×	10.94s	25.41s
Loop tiling 02			□ 21.58s

Ștefan Ciobâcă, Dorel Lucanu, and Andrei-Sebastian Buruiană. Operationally-based program equivalence proofs using LCTRSs. *J. Log. Algebraic Methods Program.*, 135:100894, 2023

Impact

- ISR 2019: invited speaker (`RMT` tool);

- co-chair WPTE 2022, WPTE 2023, SC member since 2024;

- Unification modulo builtins: José Meseguer: "The recent work of S. Ciobaca, A. Arusoaie, and D. Lucanu [22] saves the day.";

- Unification work: Dennis de Champeaux, after a 20-year retirement (Journal of Automated Reasoning 2022);

- AC Unification Library: potential integration into the Tamarin prover.

Outline

Program Verification (using Dafny)

Dafny is a verification-enabled programming language.

```
method binarySearch(a: array<int>, key : int) returns (r : int)
  requires ∀ j, k • 0 ≤ j < k < a.Length ⟹ a[j] ≤ a[k]
  ensures r ≥ 0 ⟹ 0 ≤ r < a.Length ∧ a[r] == key
  ensures r < 0 ⟹ ∀ k • 0 ≤ k < a.Length - 1 ⟹ a[k] ≠ key
{
  var left : int := 0;
  var right : int := a.Length - 1;
  while (left ≤ right)
    invariant 0 ≤ left ≤ a.Length
    invariant -1 ≤ right < a.Length
    invariant ∀ k • 0 ≤ k < left ⟹ a[k] < key
    invariant ∀ k • right < k < a.Length ⟹ a[k] > key
    decreases right - left
  {
    var mid : int := (left + right) / 2;
    if (key < a[mid]) {
      right := mid - 1;
    } else if (key > a[mid]) {
      left := mid + 1;
    } else {
      return mid;
    }
  }
  return -1;
}
```

Formalizing the CDCL Algorithm

Input: formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge \ldots$
Output: is the formula satisfiable?

2018: Proposed to Cezar-Constantin Andrici as a BSc thesis project to implement, specify and verify CDCL in Dafny.

1. **Unit propagation (or Boolean constraint propagation)**
2. **Fast data structures**.
3. **Variable ordering heuristics**.
4. **Backjumping**
5. **Conflict analysis**
6. **Clause learning and clause forgetting**
7. **Restart strategy**

Formalizing the ~~CDCL~~DPLL Algorithm

2019: We implemented, specified and verified a simplified version of DPLL in Dafny.

1. **Unit propagation (or Boolean constraint propagation)**
2. **Fast data structures**.
3. **Variable ordering heuristics**.
4. ~~Backjumping~~
5. ~~Conflict analysis~~
6. ~~Clause learning and clause forgetting~~
7. ~~Restart strategy~~

Cezar-Constantin Andrici and Ștefan Ciobâcă. Verifying the DPLL algorithm in dafny. In Mircea Marin and Adrian Crăciun, editors, *Proceedings Third Symposium on Working Formal Methods, FROM 2019, Timișoara, Romania, 3-5 September 2019*, volume 303 of *EPTCS*, pages 3–15, 2019

Cezar-Constantin Andrici and Ștefan Ciobâcă. A verified implementation of the DPLL algorithm in Dafny. *Mathematics*, 10(13), 2022

Formalizing the CNF Transformation(s)

Input: formula $\neg(x_1 \wedge x_2) \wedge \ldots$
Output: $(\neg x_1 \vee \neg x_2) \wedge \ldots$

2019: Proposed to Viorel Iordache a BSc thesis project to implement, specify and verify the **CNF transformation** in Dafny.

2020: BSc. defense (two transformations: the textbook approach and the Tseitin transformation).

> Viorel Iordache and Ștefan Ciobâcă. Verifying the conversion into CNF in Dafny. In Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*, volume 13038 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2021

2022: MSc. thesis on porting the CNF transformation to Stainless: `https://github.com/iordacheviorel/cnf-scala`.

The QOI File Format

THE QUITE OK IMAGE FORMAT

Specification **Version 1.0**, 2022.01.05 — qoiformat.org — Dominic Szablewski

A QOI file consists of a 14-byte header, followed by any number of data "chunks" and an 8-byte end marker.

```
qoi_header {
    char     magic[4];   // magic bytes "qoif"
    uint32_t width;      // image width in pixels (BE)
    uint32_t height;     // image height in pixels (BE)
    uint8_t  channels;   // 3 = RGB, 4 = RGBA
    uint8_t  colorspace; // 0 = sRGB with linear alpha
                         // 1 = all channels linear
};
```

The colorspace and channel fields are purely informative. They do not change the way data chunks are encoded.

Images are encoded row by row, left to right, top to bottom. The decoder and encoder start with **{r: 0, g: 0, b: 0, a: 255}** as the previous pixel value. An image is complete when all pixels specified by **width * height** have been covered. Pixels are encoded as:

- a run of the previous pixel
- an index into an array of previously seen pixels
- a difference to the previous pixel value in r,g,b
- full r,g,b or r,g,b,a values

The color channels are assumed to not be premultiplied with the alpha channel ("un-premultiplied alpha").

A running **array[64]** (zero-initialized) of previously seen pixel values is maintained by the encoder and decoder. Each pixel that is seen by the encoder and decoder is put into this array at the position formed by a hash function of the color value. In the encoder, if the pixel value at the index matches the current pixel, this index position is written to the stream as **QOI_OP_INDEX**. The hash function for the index is:

```
index_position = (r * 3 + g * 5 + b * 7 + a * 11) % 64
```

Each chunk starts with a 2- or 8-bit tag, followed by a number of data bits. The bit length of chunks is divisible by 8 - i.e. all chunks are byte aligned. All values encoded in these data bits have the most significant bit on the left. The 8-bit tags have precedence over the 2-bit tags. A decoder must check for the presence of an 8-bit tag first.

The byte stream's end is marked with 7 **0x00** bytes followed by a single **0x01** byte.

The possible chunks are:

QOI_OP_RGB

Byte[0]	Byte[1]	Byte[2]	Byte[3]
7 6 5 4 3 2 1 0	7 .. 0	7 .. 0	7 .. 0
1 1 1 1 1 1 1 0	red	green	blue

8-bit tag b11111110
8-bit red channel value
8-bit green channel value
8-bit blue channel value

The alpha value remains unchanged from the previous pixel.

QOI_OP_RGBA

Byte[0]	Byte[1]	Byte[2]	Byte[3]	Byte[4]
7 6 5 4 3 2 1 0	7 .. 0	7 .. 0	7 .. 0	7 .. 0
1 1 1 1 1 1 1 1	red	green	blue	alpha

8-bit tag b11111111
8-bit red channel value
8-bit green channel value
8-bit blue channel value
8-bit alpha channel value

QOI_OP_INDEX

Byte[0]
7 6 5 4 3 2 1 0
0 0 index

2-bit tag b00
6-bit index into the color index array: 0..63

A valid encoder must not issue 2 or more consecutive QOI_OP_INDEX chunks to the same index. QOI_OP_RUN should be used instead.

QOI_OP_DIFF

Byte[0]
7 6 5 4 3 2 1 0
0 1 dr dg db

2-bit tag b01
2-bit red channel difference from the previous pixel -2..1
2-bit green channel difference from the previous pixel -2..1
2-bit blue channel difference from the previous pixel -2..1

The difference to the current channel values are using a wraparound operation, so **1 - 2** will result in **255**, while **255 + 1** will result in **0**.

Values are stored as unsigned integers with a bias of **2**. E.g. **-2** is stored as **0** (**b00**). **1** is stored as **3** (**b11**).

The alpha value remains unchanged from the previous pixel.

QOI_OP_LUMA

Byte[0]	Byte[1]
7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
1 0 diff green	dr - dg db - dg

2-bit tag b10
6-bit green channel difference from the previous pixel -32..31
4-bit red channel difference minus green channel difference -8..7
4-bit blue channel difference minus green channel difference -8..7

The green channel is used to indicate the general direction of change and is encoded in 6 bits. The red and blue channels (dr and db) base their diffs off of the green channel difference. I.e.:

```
dr_dg = (cur_px.r - prev_px.r) - (cur_px.g - prev_px.g)
db_dg = (cur_px.b - prev_px.b) - (cur_px.g - prev_px.g)
```

The difference to the current channel values are using a wraparound operation, so **10 - 13** will result in **253**, while **250 + 7** will result in **1**.

Values are stored as unsigned integers with a bias of **32** for the green channel and a bias of **8** for the red and blue channel.

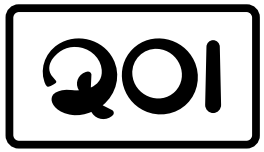
The alpha value remains unchanged from the previous pixel.

QOI_OP_RUN

Byte[0]
7 6 5 4 3 2 1 0
1 1 run

2-bit tag b11
6-bit run-length repeating the previous pixel: 1..62

The run-length is stored with a bias of **-1**. Note that the run-lengths **63** and **64** (**b111110** and **b111111**) are illegal as they are occupied by the **QOI_OP_RGB** and **QOI_OP_RGBA** tags.



THE QUITE OK IMAGE FORMAT

Specification **Version 1.0**, 2022.01.05 — qoiformat.org — Dominic Szablewski

---

A QOI file consists of a 14-byte header, followed by any number of data “chunks” and an 8-byte end marker.

```
qoi_header {
    char     magic[4];   // magic bytes "qoif"
    uint32_t width;      // image width in pixels (BE)
    uint32_t height;     // image height in pixels (BE)
    uint8_t  channels;   // 3 = RGB, 4 = RGBA
    uint8_t  colorspace; // 0 = sRGB with linear alpha
                         // 1 = all channels linear
};
```

The colorspace and channel fields are purely informative. They do not change the way data chunks are encoded.

Images are encoded row by row, left to right, top to bottom. The decoder and encoder start with **{r: 0, g: 0, b: 0, a: 255}** as the previous pixel value. An image is complete when all pixels specified by **width * height** have been covered. Pixels are encoded as:

- a run of the previous pixel
- an index into an array of previously seen pixels
- a difference to the previous pixel value in r,g,b
- full r,g,b or r,g,b,a values

The color channels are assumed to not be premultiplied with the alpha channel (“un-premultiplied alpha”).

A running **array[64]** (zero-initialized) of previously seen pixel values is maintained by the encoder and decoder. Each pixel that is seen by the encoder and decoder is put into this array at the position formed by a hash function of the color value. In the encoder, if the pixel value at the index matches the current pixel, this index position is written to the stream as **QOI_OP_INDEX**. The hash function for the index is:

```
index_position = (r * 3 + g * 5 + b * 7 + a * 11) % 64
```

QOI_OP_INDEX

Byte[0]							
7	6	5	4	3	2	1	0
0	0	index					

2-bit tag b00
6-bit index into the color index array: 0..63

A valid encoder must not issue 2 or more consecutive QOI_OP_INDEX chunks to the same index. QOI_OP_RUN should be used instead.

QOI_OP_DIFF

Byte[0]							
7	6	5	4	3	2	1	0
0	1	dr		dg		db	

2-bit tag b01
2-bit red channel difference from the previous pixel -2..1
2-bit green channel difference from the previous pixel -2..1
2-bit blue channel difference from the previous pixel -2..1

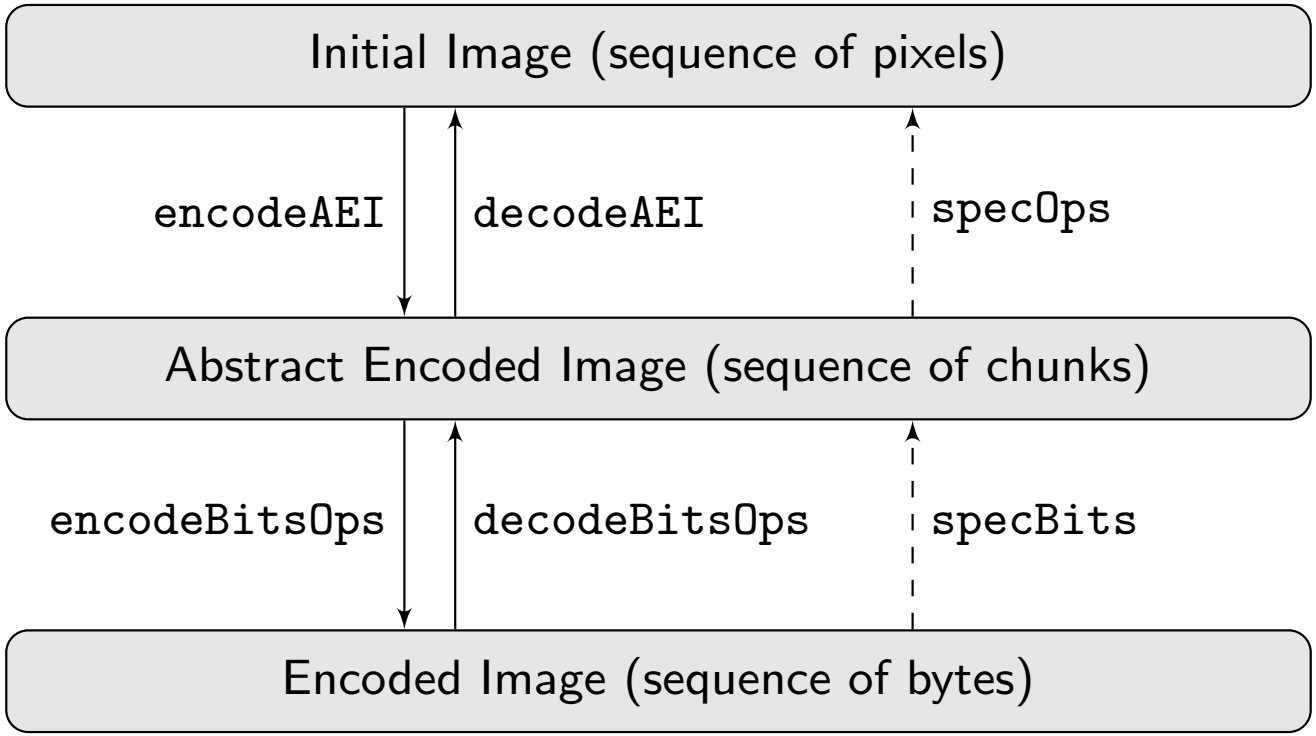
The difference to the current channel values are using a wraparound operation, so **1 - 2** will result in **255**, while **255 + 1** will result in **0**.

Values are stored as unsigned integers with a bias of **2**. E.g. **-2** is stored as **0** (**b00**). **1** is stored as **3** (**b11**).

The alpha value remains unchanged from the previous pixel.

QOI_OP_LUMA

Verifying a QOI implementation in Dafny



Ștefan Ciobâcă and Diana-Elena Gratie. Implementing, specifying, and verifying the QOI format in Dafny: A case study. In Nikolai Kosmatov and Laura Kovács, editors, *Integrated Formal Methods - 19th International Conference, IFM 2024, Manchester, UK, November 13-15, 2024, Proceedings*, volume 15234 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2024

Impact

- Verified DPLL implementation: case study at University of Manchester;

- Cezar-Constantin Andrici: PhD under the direction of Cătălin Hriţcu;

- WoLLIC 2022 organized in Iaşi (`https://wollic2022.github.io/`);

- About 13 BSc./MSc. theses on Dafny or related topics (students learned Dafny, or other systems, like F*, as part of their work on the project):
  1. (February, 2025) Alina-Adriana Haidău: *Verifying an algorithm for the discrete version of the knapsack problem in Dafny* (in Romanian);
  2. (July, 2024) Roxana Mihaela Timon: *Verifying an algorithm for the weighted activity selection problem in Dafny* (in Romanian);
  3. (July, 2024) Daniel-Antoniu Dumitru: *Implementing and Verifying the Boyer-Moore-Horspool Algorithm in F** (in English);
  4. (June-July, 2023) Alexandru Donica: *Verifying the DPLL algorithm F** (in Romanian);
  5. (June-July, 2023) Bianca-Maria Buzilă: *Computing CNFs in F*. Implementation and verification* (in Romanian);

Impact

- 2024/2025: elective lecture on *Verification-Driven Program Development*;

- Participation to VerifyThis 2024;
- VerifyThis 2025: Best Contributed Problem;

- 2025: Amazon Research Award for a project on extending Dafny with an interactive proof mode.

`https://profs.info.uaic.ro/stefan.ciobaca/aipmda.html`

Ștefan Ciobâcă (PI)

Roxana-Mihaela Timon

Andrei-Felix Similachi (alumnus)

Ștefan Mercaș

Lucian Gâdioi (admin)

Outline

Perspectives

- Lessons learned:
  - access to (old) research papers;
  - reproducibility;
  - mixing teaching and research;
  - some results I have not marketed properly;
  - new perspectives and ideas on old results.

- Future research work:
  - make it easy to develop verified programs:
    - improve proof automation/predictability,
    - provide better standard library,
    - relational verifier in Dafny;
  - verified program verifier;
  - solve the *unification modulo axiomatized symbols* problems.

- Acknowledgments: family, friends, mentors, students, co-authors, colleagues, fellow researchers.

References.

[1] Carmine Abate, Roberto Blanco, Ștefan Ciobâcă, Adrien Durier, Deepak Garg, Cătălin Hriţcu, Marco Patrignani, Éric Tanter, and Jérémy Thibault. An extended account of trace-relating compiler correctness and secure compilation. *ACM Trans. Program. Lang. Syst.*, 43(4):14:1–14:48, 2021.

[2] Cezar-Constantin Andrici and Ștefan Ciobâcă. Verifying the DPLL algorithm in dafny. In Mircea Marin and Adrian Crăciun, editors, *Proceedings Third Symposium on Working Formal Methods, FROM 2019, Timișoara, Romania, 3-5 September 2019*, volume 303 of *EPTCS*, pages 3–15, 2019.

[3] Cezar-Constantin Andrici, Ștefan Ciobâcă, Cătălin Hriţcu, Guido Martínez, Exequiel Rivas, Éric Tanter, and Théo Winterhalter. Securing verified IO programs against unverified code in F*. *Proc. ACM Program. Lang.*, 8(POPL):2226–2259, 2024.

[4] Cezar-Constantin Andrici and Ștefan Ciobâcă. A verified implementation of the DPLL algorithm in Dafny. *Mathematics*, 10(13), 2022.

[5] Andrei Arusoaie, Ștefan Ciobâcă, Vlad Craciun, Dragos Gavrilut, and Dorel Lucanu. A comparison of open-source static analysis tools for vulnerability detection in C/C++ code. In Tudor Jebelean, Viorel Negru, Dana Petcu, Daniela Zaharie, Tetsuo Ida, and Stephen M. Watt, editors, *19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2017, Timisoara, Romania, September 21-24, 2017*, pages 161–168. IEEE Computer Society, 2017.

[6] Andrei-Sebastian Buruiană and Ștefan Ciobâcă. Reducing total correctness to partial correctness by a transformation of the language semantics. In Joachim

Niehren and David Sabel, editors, *Proceedings Fifth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2018, Oxford, England, 8th July 2018*, volume 289 of *EPTCS*, pages 1–16, 2018.

[7] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23, 2016.

[8] Ştefan Ciobâcă. From small-step semantics to big-step semantics, automatically. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2013.

[9] Ştefan Ciobâcă. Reducing partial equivalence to partial correctness. In Franz Winkler, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, Timisoara, Romania, September 22-25, 2014*, pages 164–171. IEEE Computer Society, 2014.

[10] Ştefan Ciobâcă, Andrei Arusoaie, and Dorel Lucanu. Unification modulo builtins. In Lawrence S. Moss, Ruy J. G. B. de Queiroz, and Maricarmen Martínez, editors, *Logic, Language, Information, and Computation - 25th International Workshop, WoLLIC 2018, Bogota, Colombia, July 24-27, 2018, Proceedings*, volume 10944 of *Lecture Notes in Computer Science*, pages 179–195. Springer, 2018.

[11] Ştefan Ciobâcă and Diana-Elena Gratie. Implementing, specifying, and verifying the QOI format in Dafny: A case study. In Nikolai Kosmatov and Laura Kovács, editors, *Integrated Formal Methods - 19th International Conference, IFM 2024, Manchester,*

UK, November 13-15, 2024, Proceedings, volume 15234 of *Lecture Notes in Computer Science*, pages 35–52. Springer, 2024.

[12] Ștefan Ciobâcă and Dorel Lucanu. A coinductive approach to proving reachability properties in logically constrained term rewriting systems. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2018.

[13] Ștefan Ciobâcă, Dorel Lucanu, and Andrei-Sebastian Buruiană. Operationally-based program equivalence proofs using LCTRSs. *J. Log. Algebraic Methods Program.*, 135:100894, 2023.

[14] Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A language-independent proof system for mutual program equivalence. In Stephan Merz and Jun Pang, editors, *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg, Luxembourg, November 3-5, 2014. Proceedings*, volume 8829 of *Lecture Notes in Computer Science*, pages 75–90. Springer, 2014.

[15] Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A theoretical foundation for programming languages aggregation. In Mihai Codescu, Razvan Diaconescu, and Ionut Tutu, editors, *Recent Trends in Algebraic Development Techniques - 22nd International Workshop, WADT 2014, Sinaia, Romania, September 4-7, 2014, Revised Selected Papers*, volume 9463 of *Lecture Notes in Computer Science*, pages 30–47. Springer, 2014.

[16] Ștefan Ciobâcă, Dorel Lucanu, Vlad Rusu, and Grigore Roșu. A language-independent proof system for full program equivalence. *Formal Aspects Comput.*, 28(3):469–497, 2016.

[17] Viorel Iordache and Ștefan Ciobâcă. Verifying the conversion into CNF in Dafny. In Alexandra Silva, Renata Wassermann, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 27th International Workshop, WoLLIC 2021, Virtual Event, October 5-8, 2021, Proceedings*, volume 13038 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2021.

[18] Cynthia Kop and Naoki Nishida. Term rewriting with logical constraints. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013.

[19] Valeriu Motroi and Ștefan Ciobâcă. A typo in the Paterson-Wegman-de Champeaux algorithm. *CoRR*, abs/2007.00304, 2020.

[20] Valeriu Motroi and Ștefan Ciobâcă. A note on the performance of algorithms for solving linear Diophantine equations in the naturals. *CoRR*, abs/2104.05200, 2021.

[21] Grigore Roșu and Andrei Ștefănescu. Checking reachability using matching logic. In Gary T. Leavens and Matthew B. Dwyer, editors, *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012*, pages 555–574. ACM, 2012.

[22] Grigore Roșu, Andrei Ștefănescu, Ștefan Ciobâcă, and Brandon M. Moore. One-path reachability logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 358–367. IEEE Computer Society, 2013.

[23] Shinichi Shiraishi, Veena Mohan, and Hemalatha Marimuthu. Test suites for benchmarks of static analysis tools. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 12–15, 2015.

[24] Andrei Ștefănescu, Ștefan Ciobâcă, Radu Mereuță, Brandon M. Moore, Traian-Florin Șerbănuță, and Grigore Roșu. All-path reachability logic. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8560 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2014.