Overview of Cooperative Verification Approaches

A Pilot Survey

Dirk Beyer and Sudeep Kanav

LMU Munich, Germany

Cooperative Verification

Beyer and Wehrheim define cooperative verification as:

> ... ***cooperative verifiers provide exchangeable information (verification artifacts) to other verifiers or consume such information from other verifiers*** ... *[10]*

Cooperative Verification

Key points:

- verifiers: model checkers, test generators, theorem provers
- cooperative: produce and/or consume knowledge about the verification task other than the final outcome
- knowledge exchange: the knowledge is shared by the means of verification artifacts

Cooperative Verification

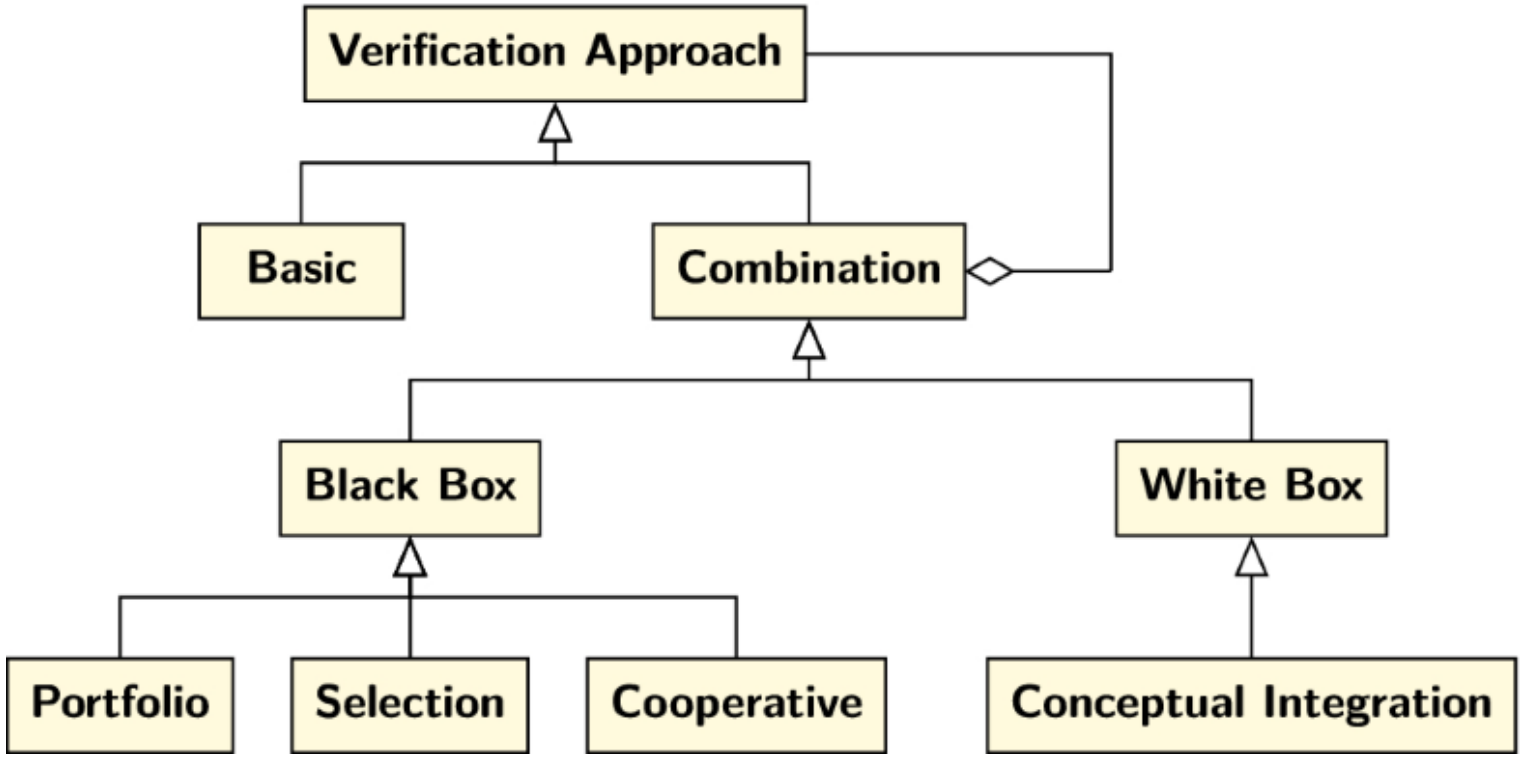

Figure: Hierarchy of verification approaches [10]

Cooperative Verification

A verification approach that is:

- combination
- black-box
- neither portfolio, nor selection

Portfolio

Portfolio is not cooperative because:

- each verifier is working in isolation
- no exchange of artifacts, i.e., the verifiers do not benefit from the work done by other verifiers

Selection

Selection is not cooperative because:

- communicating tools are not verifiers
- a selector selects a verifier; then that verifier is executed

Cooperative Verification

Definition of cooperative verification:

- **Black-box**: it is a black-box combination
- **Cooperating Analyzers**: the principal cooperating tools are analyzers (model checkers, theorem provers, or test generators)
  at least two cooperating analyzers[1]
- **Artifact Exchange**: the cooperating analyzers produce and / or consume verification artifacts (produced by other cooperating verifiers)

[1] tools that produce new knowledge about the verification task

Classification

Classification

- conditional verification
- refinement of the verification result
- iterative
- *towards* service oriented

Conditional Verification - Examples

Title	Year	Black-Box	Cooperating Analyzers	Artifact Exchange
Conditional Model Checking [4]	2012	✓	✓	✓
Collaborative Verification and Testing with Explicit Assumptions [11]	2012	✓	✓	✓
Just test what you cannot verify! [12]	2015	✓	✓	✓
CondTest [7]	2019	✓	✓	✓

Refinement of the Verification Result - Examples

Title	Year	Black-Box	Cooperating Analyzers	Artifact Exchange
Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples [15]	2012	✓	✓	✓
Stepwise Testification [1]	2015	✓	✓	✓

Iterative - Examples

Title	Year	Black-Box	Cooperating Analyzers	Artifact Exchange
CondTest [7]	2019	✓	✓	✓
CoVEGI [14]	2021	✓	✓	✓
Modular CEGAR [3]	2022	✓	✓	✓

Towards Service Oriented - Example

Title	Year	Black-Box	Cooperating Analyzers	Artifact Exchange
CoVEGI [14]	2021	✓	✓	✓

But, our definition leaves out cases which we thought were cooperative!

Exclusions - Reduction Techniques

Examples of excluded techniques:

- Reducer based construction of conditional model checking [6]
- MetaVal [9]

Exclusions - Reduction Techniques

Examples of excluded techniques:

- Reducer based construction of conditional model checking [6]
- MetaVal [9]

Only one of the tools in the combination is a verifier. Moreover, these are not cooperating verifiers.

Exclusions - Reduction Techniques

Table: Examples of Excluded Approaches

Title	Year	Black Box	Cooperating Analyzers	Artifact Exchange
Reducer based CMC [6]	2012	✓	✗	✓
MetaVal [9]	2020	✓	✗	✓

Some Unclear Cases - CoVeriTest

Is CoVeriTest [5] cooperative?

- the article explicitly states that it is cooperative
- but it is not black-box
- the cooperating artifact is not exported

Some Unclear Cases - CoVeriTest

Is CoVeriTest [5] cooperative?

- the article explicitly states that it is cooperative
- but it is not black-box
- the cooperating artifact is not exported

It is not cooperative according to our definition.

Some Unclear Cases - k-induction

Is "Boosting k-induction with continuously refined invariants" [2] cooperative?

- it is not black-box
- the cooperating artifact is not exported

It is not cooperative according to our definition.

Some Unclear Cases - Reuse in Regression

The verification task changes in regression verification, but the artifacts can be reused. Examples:

- precision reuse [8]
- test-suite augmentation

Some Unclear Cases - Precision Reuse

Is precision reuse [8] cooperative?

- a black-box technique
- artifact is exported
- but a regression technique, i.e., the verification task changes

Rethinking the Definition

Maybe our definition of cooperative verification is too restrictive! We simply wanted to say:

- it should be black box
- some (at least two) tools in the combination should produce and share (or at least make use of existing) some knowledge about the verification task
- tools in the combination should export and import the artifacts

Shall we weaken our definition?

Cooperative Verification

An approach is called **cooperative verification**, if

- *identifiable actors* pass information in form of
- *identifiable artifacts* towards the common objective of
- solving a verification problem,

where at least two of these actors are analyzers.

Cooperative Verification

An approach is called **cooperative verification**, if

- *identifiable actors* pass information in form of
- *identifiable artifacts* towards the common objective of
- solving a verification problem,

where at least two of these actors are analyzers.

Examples:

- Identifiable actor: off-the-shelf components, binaries, agents, web services
- Identifiable artifacts: programs, witnesses, ARGs, test suites
- verification problem: verification task, test task, feasibility check, refinement

Cooperative Techniques - Updated Definition

Title	Year	Identifiable Actors	Identifiable Actors	Cooperating Analyzers
Conditional Model Checking [4]	2012	✓	✓	✓
Collaborative Verification and Testing with Explicit Assumptions [11]	2012	✓	✓	✓
Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples [15]	2012	✓	✓	✓
Just test what you cannot verify! [12]	2015	✓	✓	✓
Stepwise Testification [1]	2015	✓	✓	✓
CondTest [7]	2019	✓	✓	✓
CoVeriTest [5]	2019	✓	✓	✓
CoVEGI [14]	2021	✓	✓	✓
Modular CEGAR [3]	2022	✓	✓	✓

References I

[1] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). doi:10.1145/2786805.2786867

[2] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015). doi:10.1007/978-3-319-21690-4_42

[3] Beyer, D., Haltermann, J., Lemberger, T., Wehrheim, H.: Decomposing Software Verification into Off-the-Shelf Components: An Application to CEGAR. In: Proc. ICSE. ACM (2022)

[4] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). doi:10.1145/2393596.2393664

References II

[5] Beyer, D., Jakobs, M.C.: CoVeriTest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). doi:10.1007/978-3-030-16722-6_23

[6] Beyer, D., Jakobs, M.C., Lemberger, T., Wehrheim, H.: Reducer-based construction of conditional verifiers. In: Proc. ICSE. pp. 1182–1193. ACM (2018). doi:10.1145/3180155.3180259

[7] Beyer, D., Lemberger, T.: Conditional testing: Off-the-shelf combination of test-case generators. In: Proc. ATVA. pp. 189–208. LNCS 11781, Springer (2019). doi:10.1007/978-3-030-31784-3_11

[8] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). doi:10.1145/2491411.2491429

[9] Beyer, D., Spiessl, M.: MetaVal: Witness validation via verification. In: Proc. CAV. pp. 165–177. LNCS 12225, Springer (2020). doi:10.1007/978-3-030-53291-8_10

References III

[10] Beyer, D., Wehrheim, H.: Verification artifacts in cooperative verification: Survey and unifying component framework. In: Proc. ISoLA (1). pp. 143–167. LNCS 12476, Springer (2020). doi:10.1007/978-3-030-61362-4_8

[11] Christakis, M., Müller, P., Wüstholz, V.: Collaborative verification and testing with explicit assumptions. In: Proc. FM. pp. 132–146. LNCS 7436, Springer (2012). doi:10.1007/978-3-642-32759-9_13

[12] Czech, M., Jakobs, M., Wehrheim, H.: Just test what you cannot verify! In: Proc. FASE. pp. 100–114. LNCS 9033, Springer (2015). doi:10.1007/978-3-662-46675-9_7

[13] Gennari, J., Gurfinkel, A., Kahsai, T., Navas, J.A., Schwartz, E.J.: Executable counterexamples in software model checking. In: Proc. VSTTE. pp. 17–37. LNCS 11294, Springer (2018). doi:10.1007/978-3-030-03592-1_2

References IV

[14] Haltermann, J., Wehrheim, H.: CoVEGI: Cooperative verification via externally generated invariants. In: Proc. FASE. pp. 108–129. LNCS 12649 (2021). doi:10.1007/978-3-030-71500-7_6

[15] Rocha, H.O., Barreto, R.S., Cordeiro, L.C., Neto, A.D.: Understanding programming bugs in ANSI-C software using bounded model checking counter-examples. In: Proc. IFM. pp. 128–142. LNCS 7321, Springer (2012). doi:10.1007/978-3-642-30729-4_10