An Interface Theory for Program Verification

Dirk Beyer and Sudeep Kanav

LMU Munich, Germany
Motivation

What  A compositional viewpoint of software verification

Why  Decompose software verification into parts

Example  Counterexample-guided abstraction refinement (CEGAR)
Verification Interfaces

Verification interfaces are descriptions of behavior that occurs in a program:

- Program Interface
- Specification Interface
- Correctness Interface
- Violation Interface
Given a program $P$ and a specification $\phi$, verification is the problem of finding either a correctness proof for $I_P \preceq I_{\phi}$ or a violation proof for $I_P \not\preceq I_{\phi}$.\(^1\)

---

\(^1\)There are various ways for reasoning in order to obtain a proof, for example, strongest post-conditions [9] are traditionally used for correctness proofs and incorrectness logic [10] was proposed for violation proofs.
Space of Verification Interfaces (Fig. 3 in [6])
Theorems ...

- Refinement preserves correctness
- Abstraction preserves violation
- Substitutivity of interfaces
Verification

Interface Domain

- Correctness Interface ($I_C$)
- Program Interface ($I_P$)
- Violation Interface ($I_V$)

Programmer Domain

- Specification ($\phi$)
- Correctness Witness ($W_C$)
- Program ($P$)
- Violation Witness ($W_V$)
- Test Vector ($T$)

Transformations:
- $I_C$ to $W_C$ (transformation)
- $I_P$ to $P$ (transformation)
- $I_V$ to $W_V$ (transformation)
- $T$ to $W_V$ (represents)

Relationships:
- $I_C$ satisfies $\phi$
- $I_P$ to $P$ (transformation)
- $I_V$ to $W_V$ (transformation)
- $T$ to $W_V$ (represents)

Refinements:
- $I_C$ to $I_P$ (refinement)
- $I_V$ to $I_P$ (refinement)
- $T$ to $I_V$ (represents)

Summarizations:
- $W_C$ to $\phi$ (summarization)
- $P$ to $W_V$ (summarization)
- $W_V$ to $T$ (summarization)
Result Validation (Fig. 10 in [6])

Interface Domain

- Correctness Interface ($I_C$)
- Program Interface ($I_P$)
- Violation Interface ($I_V$)

Programmer Domain

- Specification ($\phi$)
- Program ($P$)
- Violation Witness ($W_V$)
- Test Vector ($T$)

- Correctness Witness ($W_C$)

Transformations:
- $I_C \rightarrow W_C$
- $I_P \rightarrow P$
- $I_V \rightarrow W_V$

Abstracts:
- $I_C \leftarrow \text{abstracts}$
- $I_P \leftarrow \text{refines}$
- $I_V \leftarrow \text{represents}$

Satisfies:
- $I_C \rightarrow \phi$
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$

Refines:
- $I_P \rightarrow P$
- $T \rightarrow \text{represents}$

Concretization:
- $W_C \rightarrow \text{summarization}$
- $P \rightarrow \text{concretization}$

Testifies:
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$
- $T \rightarrow \text{testifies}$

Abstracts:
- $I_C \leftarrow \text{abstracts}$
- $I_P \leftarrow \text{refines}$
- $I_V \leftarrow \text{represents}$

Satisfies:
- $I_C \rightarrow \phi$
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$

Refines:
- $I_P \rightarrow P$
- $T \rightarrow \text{represents}$

Concretization:
- $W_C \rightarrow \text{summarization}$
- $P \rightarrow \text{concretization}$

Testifies:
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$
- $T \rightarrow \text{testifies}$

Abstracts:
- $I_C \leftarrow \text{abstracts}$
- $I_P \leftarrow \text{refines}$
- $I_V \leftarrow \text{represents}$

Satisfies:
- $I_C \rightarrow \phi$
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$

Refines:
- $I_P \rightarrow P$
- $T \rightarrow \text{represents}$

Concretization:
- $W_C \rightarrow \text{summarization}$
- $P \rightarrow \text{concretization}$

Testifies:
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$
- $T \rightarrow \text{testifies}$

Abstracts:
- $I_C \leftarrow \text{abstracts}$
- $I_P \leftarrow \text{refines}$
- $I_V \leftarrow \text{represents}$

Satisfies:
- $I_C \rightarrow \phi$
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$

Refines:
- $I_P \rightarrow P$
- $T \rightarrow \text{represents}$

Concretization:
- $W_C \rightarrow \text{summarization}$
- $P \rightarrow \text{concretization}$

Testifies:
- $W_C \rightarrow \text{testifies}$
- $W_V \rightarrow \text{testifies}$
- $T \rightarrow \text{testifies}$
Decompose Verification Tools

- Interface synthesizers, to construct an interface
- Refinement checkers, to check $I_1 \preceq I_2$
- Specification checkers, to check $I \models \phi$
1. construct an abstract model $I_0$
2. check $I_0 \models \phi$; if it holds, terminate with answer (TRUE, $W_C$) (the interface $I_0$ corresponds to an interface $I_C$ in Fig. 3, the correctness witness $W_C$ in Fig. 10 is an abstraction of $I_C$)
3. extract counterexample interface $I_1$ from $I_0$ (interface $I_0$ corresponds to interface $I_{\neg C}$ in Fig. 3)
4. check $I_1 \not\models \phi$; if it holds, terminate with answer (FALSE, $W_V$) (the interface $I_1$ corresponds to an interface $I_V$ in Fig. 3, the violation witness $W_V$ in Fig. 10 is an abstraction of $I_V$)
5. extract new facts (derived from the infeasibility of $I_1$) and continue with step (1); (the interface $I_1$ corresponds to an interface $I_{\neg V}$ in Fig. 3)
CEGAR Using Verification Interfaces

\[ I_T \]

\[ I_\phi \]

\[ I_C \]

\[ I_P \]

\[ I_V \]

\[ I_T \]

\[ I_\perp \]

\[ \models \phi \]

\[ \not \models \phi \]

\[ \text{refine} \]

\[ \text{abstract} \]

\[ I_{-C_1} \]

\[ I_{-V_1} \]

\[ I_{-C_2} \]

\[ I_{-C_3} \]

\[ I_{-V_1} \]

Dirk Beyer and Sudeep Kanav
LMU Munich, Germany
Conclusion

- Unified viewpoint
- Decompose monolithic approaches
- Off-the-shelf combinations
- Document verification results using witnesses
References I

[1] de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proc. FSE. pp. 109–120. ACM (2001). doi:10.1145/503271.503226

[2] Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web service interfaces. In: Proc. WWW. pp. 148–159. ACM (2005). doi:10.1145/1060745.1060770

[3] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). doi:10.1145/2950290.2950351

[4] 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

[5] Beyer, D., Henzinger, T.A., Singh, V.: Algorithms for interface synthesis. In: Proc. CAV. pp. 4–19. LNCS 4590, Springer (2007). doi:10.1007/978-3-540-73368-3_4
[6] Beyer, D., Kanav, S.: An interface theory for program verification. In: Proc. ISoLA (1). pp. 168–186. LNCS 12476, Springer (2020). doi:10.1007/978-3-030-61362-4_9

[7] Beyer, D., Wehrheim, H.: Verification artifacts in cooperative verification: Survey and unifying component framework. arXiv/CoRR 1905(08505) (May 2019), https://arxiv.org/abs/1905.08505

[8] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003). doi:10.1145/876638.876643

[9] Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969). doi:10.1145/363235.363259

[10] O’Hearn, P.W.: Incorrectness logic. Proc. ACM Program. Lang. 4(POPL) (2020). doi:10.1145/3371078
[11] Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. 3(1), 30–50 (2000). doi:10.1145/353323.353382