Generating Sharper and Simpler Nonlinear Interpolants for Program Verification

Takamasa Okudono[1], Yuki Nishida[2], Kensuke Kojima[2], Kohei Suenaga[2], Kengo Kido[1], and Ichiro Hasuo[3]

[1]University of Tokyo, Japan; [2]Kyoto University, Japan; [3]National Institute of Informatics, Japan

Summary

Interpolation of jointly infeasible predicates plays important roles in various program verification techniques such as invariant synthesis and CEGAR. Intrigued by the recent result by Dai et al. [2] that combines real algebraic geometry and SDP optimization in synthesis of polynomial interpolants, the current paper contributes its enhancement that yields *sharper* and *simpler* interpolants. The enhancement is made possible by: theoretical observations in real algebraic geometry; and our continued fraction-based algorithm that rounds off (potentially erroneous) numerical solutions of SDP solvers. Experiment results support our tool's effectiveness; we also demonstrate the benefit of sharp and simple interpolants in program verification examples.

Motivation

- An *interpolant* I of formulas A and B that satisfy $\models \neg(A \wedge B)$ satisfying (1) $\models A \to I$, (2) $\models \neg(I \wedge B)$ and (3) I contains only variables that appear both in A and B.
- A *polynomial interpolant* is a variant of interpolants for the theory of polynomials.
  - Atomic propositions are equalities ($=$), inequalities ($\geq$) and strict inequalities ($>$) of polynomials.
  - For example, a formula $I = (y > 0)$ is an interpolant of $A = (y > x \wedge y > -x)$ and $B = (y \leq 0)$ (See Figure 1, Input 1).
- Polynomial interpolants are potentially useful for **the verification of polynomial programs** (imperative programs that support expressions with finitely many additions and multiplications).
  - Interpolants play an important role in *CEGAR* [1] (a method of predicate abstraction and program verification)
- We found that the lack of two properties below is an obstacle of the verification in the manner of CEGAR with polynomial interpolants:
  - Sharpness:
    - An interpolant I of A and B is *sharp* $\iff$ the region of A and B are "touching".
    - For example, $A = (y > x \wedge y > -x)$ and $B = (y \leq 0)$ in Figure 1, Input 1 are "touching".
    - The method of Dai et al. [2] cannot generate any sharp interpolants.
  - Simplicity:
    - An interpolant is simple if it can be expressed with coefficients of fewer digits.
    - For example, $\mathrm{xa} + 2\mathrm{ya} \geq 0$ is simpler than $1.86858 \times 10^{-10} + 54.1800\mathrm{xa} + 108.3601\mathrm{ya} \geq 0$.
    - The method of Dai et al. [2] tends to yield less simple interpolants.

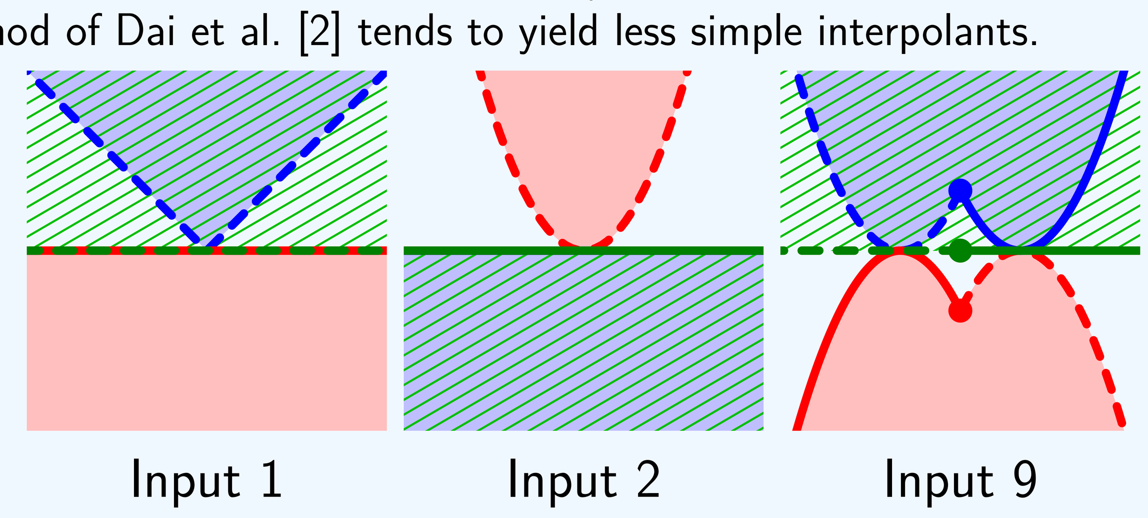

Figure 1: Experiments. The blue, orange and green areas are for A, B, I, respectively.

Method for Sharpness

- The workflow is shown in Figure 2.
- The conversion from (1) to (2) in Figure 2 is based on this proposition:

Let $\mathcal{T}$ and $\mathcal{T}'$ be

$$\begin{aligned}\mathcal{T} &= \begin{pmatrix} f_1(\vec{X},\vec{Y}) \geq 0 \ , \ \dots, \ f_s(\vec{X},\vec{Y}) \geq 0 \ , & g_1(\vec{X},\vec{Y}) > 0 \ , \ \dots, \ g_t(\vec{X},\vec{Y}) > 0 \ , \\ h_1(\vec{X},\vec{Y}) = 0 \ , \ \dots, \ h_u(\vec{X},\vec{Y}) = 0 & \end{pmatrix}, \\ \mathcal{T}' &= \begin{pmatrix} f'_1(\vec{X},\vec{Z}) \geq 0 \ , \ \dots, \ f'_{s'}(\vec{X},\vec{Z}) \geq 0 \ , & g'_1(\vec{X},\vec{Z}) > 0 \ , \ \dots, \ g'_{t'}(\vec{X},\vec{Z}) > 0 \ , \\ h'_1(\vec{X},\vec{Z}) = 0 \ , \ \dots, \ h'_{u'}(\vec{X},\vec{Z}) = 0 & \end{pmatrix}\end{aligned} \tag{1}$$

where $\vec{X}$ denotes the variables that occur in both of $\mathcal{T}, \mathcal{T}'$.

Assume there exist

$$\begin{gathered} f \in \mathcal{C}(f_1,\dots,f_s,g_1,\dots,g_t) \ , \quad f' \in \mathcal{C}(f'_1,\dots,f'_{s'},g'_1,\dots,g'_{t'}) \ , \\ g \in \mathcal{SC}(g_1,\dots,g_t) \ , \quad h \in \mathcal{I}(h_1,\dots,h_u) \ , \quad \text{and} \quad h' \in \mathcal{I}(h'_1,\dots,h'_{u'}) \\ \text{such that} \quad f + f' + g + h + h' = 0 \ . \end{gathered} \tag{2}$$

Then $\mathcal{T}$ and $\mathcal{T}'$ are disjoint. Moreover $\mathcal{S} := (f + g + h > 0)$ satisfies the conditions of an interpolant of $\mathcal{T}$ and $\mathcal{T}'$, except for Condition 3.

- Algebraic structure $\mathcal{C}$, $\mathcal{SC}$ and $\mathcal{I}$ are cones, *strict cones*, and ideals, respectively.
  - A *strict cone* $S \subset \mathbb{R}[\vec{X},\vec{Y},\vec{Z}]$ is a subset of the polynomial ring that satisfies $[f, g \in S \implies (f + g \in S \wedge fg \in S)]$ and $\mathbb{R}_{>0} \subset S$.
- Using this proposition enables us to generate sharp interpolants, which was impossible in the method of Dai et al. [2]

Method for Sharpness (Cont'd)

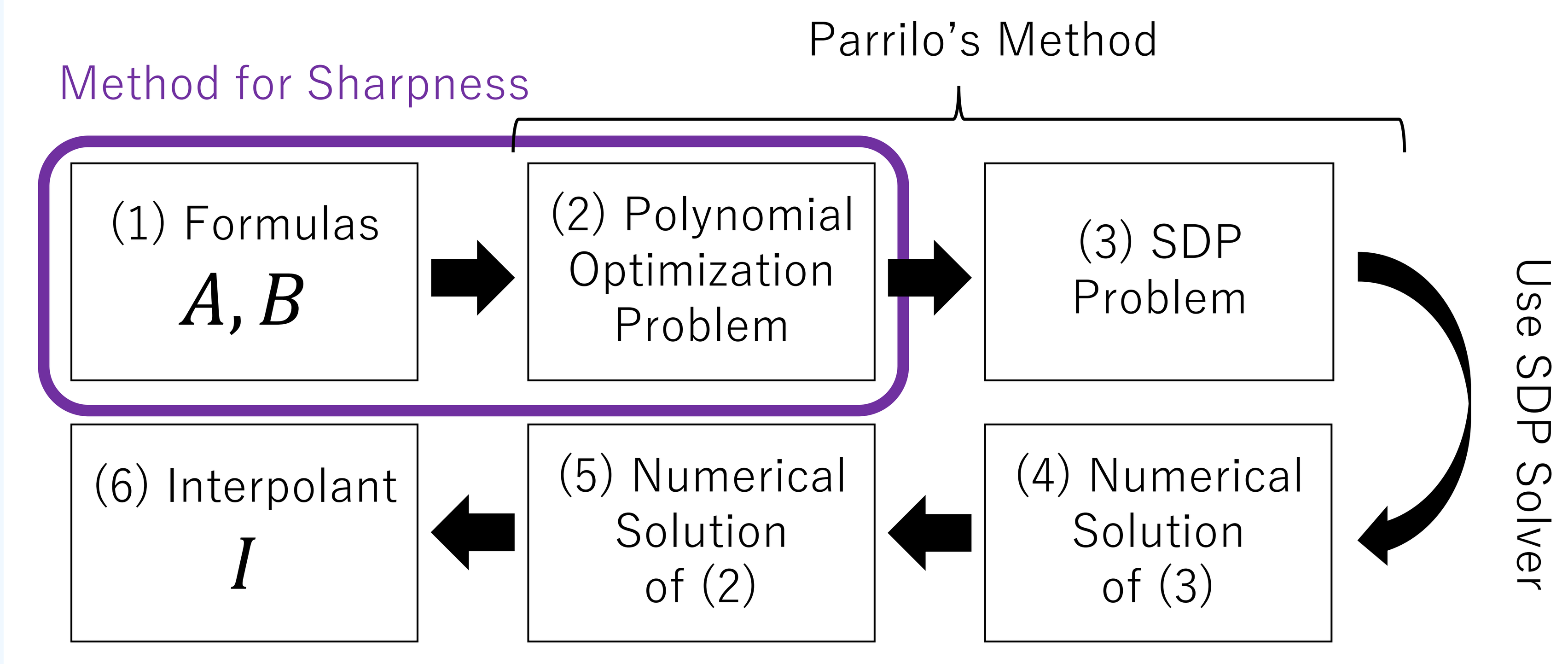

Figure 2: The Workflow of the Method for Sharpness

Method for Simplicity

- The workflow is shown in Figure 3.
- Modifying "Method for Sharpness" to get the *simplicity*.
- Our main idea is "Assume that SDP solvers return a complicated solution that is shifted from a simple solution by numerical error, and **guess the original simple solution** by simplifying the complicated solution."
- We use our *simplification-of-ratio algorithm* in the step (5) at Figure 3.
  - Our simplification-of-ratio algorithm takes depth parameter d and a ratio $a_1 : \cdots : a_n$ then returns the d-th simplified ratio $a'_1 : \dots, a'_n$.
  - By increasing d, the simplified ratio $a'_1 : \dots, a'_n$ gets faithful to the original ratio $a_1 : \cdots : a_n$ and loses simplicity.
  - For example, the simplifications of $(46.7375 : 155.0975 : 60.1733)$ are $(1 : 3 : 1) \to (3 : 10 : 4) \to (31 : 103 : 40) \to (97 : 322 : 125) \to \dots$.
  - The ratio of the coefficients of polynomials is sufficient to determine a formula, so we do not simplify the coefficients itself, but simplify the ratio (Vapnik's principle).
- The validity of the simplified solution is checked at (6) in Figure 3, so the generated interpolant at (8) is **guaranteed to be a valid interpolant**.
  - The method of Dai et al. [2] does not hold this property, so the generated interpolant could be spurious.l
- This is a best-effort method. That means, the validation at (5) in Figure 3 might fail and the method might return no interpolants even if there exists an interpolant.

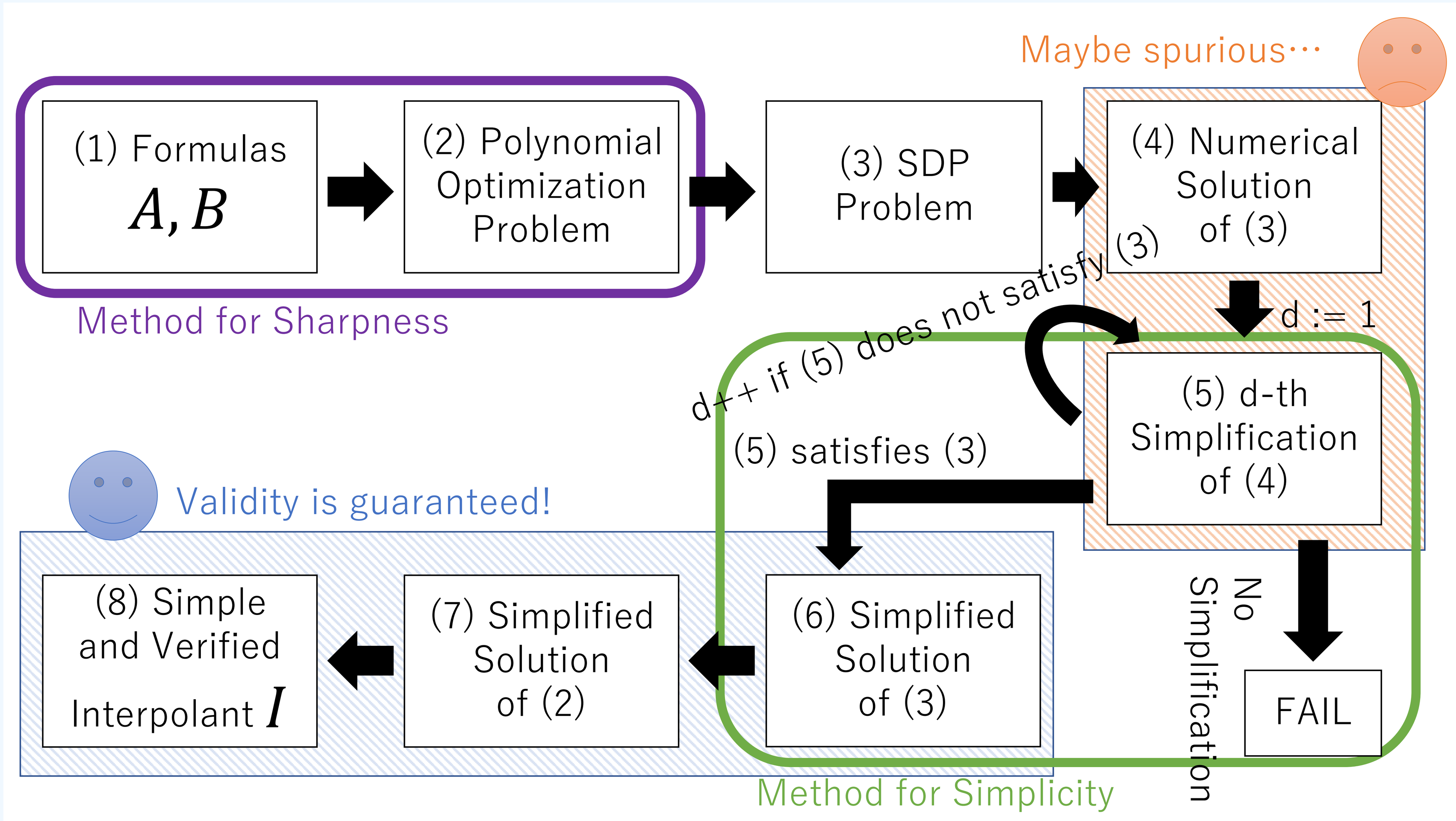

Figure 3: The Workflow of the Method for Sharpness and Simplicity

References

- Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.
  Counterexample-guided abstraction refinement for symbolic model checking.
  J. ACM, 50(5):752–794, 2003.
- Liyun Dai, Bican Xia, and Naijun Zhan.
  Generating non-linear interpolants by semidefinite programming.
  In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2013.

Acknowledgement

Thanks are due to Eugenia Sironi and Gidon Ernst for their useful comments. T.O., K. Kido and I.H. are supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), and JSPS Grants-in-Aid No. 15KT0012 & 15K11984. K. Kojima is supported by JST CREST. K.S. is supported by JST PRESTO No. JPMJPR15E5 and JSPS Grants-in-Aid No. 15KT0012. K. Kido is supported by JSPS Grant-in-Aid for JSPS Research Fellows No. 15J05580.