Higher-Order Weakest Precondition Transformers via a CPS Transformation (Extended Abstract)

Satoshi Kura

Weakest preconditions are a useful notion for program verification. Category-theoretic generalisations are studied to define weakest preconditions for various computational effects and various properties. However, those categorical frameworks often lack a syntactic aspect, namely, how to compute generic weakest preconditions syntactically. In this paper, we provide a general framework for syntactic computation of weakest preconditions. Specifically, we prove that a CPS transformation is a syntactic counterpart of semantic weakest preconditions in a general situation. We also provide several instances of our framework to show that our framework can cover various problems of program verification. Notably, we can reproduce results from two existing papers as instances of our framework, and we can also apply our framework to a new problem.

Weakest Preconditions and Program Verification

Weakest preconditions [4] are useful notion for program verification and enable us to generate verification conditions (or logical constraints) from a given pair of a program and a specification. Since such verification conditions are logical formulas that do not depend on the syntax of programming languages, checking the validity of verification conditions is usually easier than the original verification problem. Based on this idea, program verification based on weakest preconditions is implemented in e.g. Why3 [5] and Boogie [3].

Generic Weakest Preconditions

There are many variations of weakest preconditions such as total and partial correctness for programs that may not terminate, may and must correctness for nondeterministic programs, and the weakest pre-expectation [8] and the expected runtime transformer [6] for probabilistic programs. Category-theoretic generalisation of weakest preconditions is also studied since it is well-known that computational effects such as non-termination, nondeterminism, and probabilistic nondeterministic are uniformly captured by monads. For example, the work [1] gives a general framework using monads, which subsumes various weakest preconditions including the examples above.

Since monads give semantics of functional programs with computational effects, the work [1] naturally gives rise to a *semantic* definition of weakest preconditions for such programs. However, the semantic definition is not convenient for program verification because we often want to *syntactically* compute a formula that represents the weakest precondition. Syntactic computation of weakest preconditions has been mainly studied for imperative programs, and there are no existing work on syntactic and generic weakest preconditions for functional programs with computational effects.

Our Results

In this talk, we consider functional programs with computational effects and recursion, and prove that generic weakest preconditions can be syntactically obtained by a CPS transformation if we use a postcondition as a continuation. We define a CPS transformation as a syntactic translation from a source programming language to a target lan-

CPS 変換による高階最弱事前条件変換子 (Extended Abstract)

内藏 理史, 国立情報学研究所, National Institute of Informatics.

guage of higher-order modal fixed-point logic where a type of proposition **Prop** is used as an answer type. Given a well-typed term $x : \tau \vdash M : \rho$, the CPS transformation $(-)^\gamma$ gives a well-typed formula $x : \tau^\gamma \vdash M^\gamma : (\rho^\gamma \to \mathbf{Prop}) \to \mathbf{Prop}$. Assume $\tau^\gamma = \tau$ and $\rho^\gamma = \rho$, which is the case if τ and ρ do not contain function types. Now, the type of continuation $\rho \to \mathbf{Prop}$ is the same as the type of postcondition, and M^γ can be regarded as a function that takes a postcondition of type $\rho \to \mathbf{Prop}$ and returns a precondition of type $\tau \to \mathbf{Prop}$. Our main theorem states that M^γ actually gives a syntactic counterpart of the weakest precondition defined in [1].

Our main theorem enables syntactic computations of weakest preconditions. Here, we emphasize that we can apply our main theorem to programs with computational effects such as nondeterministic or probabilistic programs because our framework inherits the generality of [1]. We have several instances of our main theorem listed below.

- We have instances for total and partial correctness. This is the simplest situation since we do not consider any computational effect except for non-termination caused by infinite loops.
- We reproduce two existing works [7] [2] of program verification using our main theorem. The first one [7] is about problems of trace properties and may-/must-reachability. Given a nondeterministic program with output operations, these problems ask whether any output satisfies a certain property. The second one [2] is about expected cost analyses. Given a probabilistic program with a notion of cost, the aim is to estimate the expected cost of the program.
- Our main theorem not only generalises two existing works but can also be used to obtain new instances. For example, we extend expected cost analyses [2] by (1) allowing continuous distributions and (2) enabling analysis of higher moments of cost, which leads to a new instance of cost moment analysis.

These instances exemplify the potential of our general framework for the study of program verification.

References

[1] Aguirre, A. and Katsumata, S.-y.: Weakest Preconditions in Fibrations, *Electronic Notes in Theoretical Computer Science*, Vol. 352(2020), pp. 5–27.

[2] Avanzini, M., Barthe, G., and Dal Lago, U.: On Continuation-Passing Transformations and Expected Cost Analysis, *Proceedings of the ACM on Programming Languages*, Vol. 5, No. ICFP(2021), pp. 1–30.

[3] Barnett, M., Chang, B.-Y. E., DeLine, R., Jacobs, B., and Leino, K. R. M.: Boogie: A Modular Reusable Verifier for Object-Oriented Programs, *Formal Methods for Components and Objects*, Vol. 4111, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 364–387.

[4] Dijkstra, E. W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs, *Communications of the ACM*, Vol. 18, No. 8(1975), pp. 453–457.

[5] Filliâtre, J.-C. and Paskevich, A.: Why3 — Where Programs Meet Provers, *Programming Languages and Systems*, Vol. 7792, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 125–128.

[6] Kaminski, B. L., Katoen, J.-P., Matheja, C., and Olmedo, F.: Weakest Precondition Reasoning for Expected Runtimes of Randomized Algorithms, *Journal of the ACM*, Vol. 65, No. 5(2018), pp. 1–68.

[7] Kobayashi, N., Tsukada, T., and Watanabe, K.: Higher-Order Program Verification via HFL Model Checking, *Programming Languages and Systems*, Vol. 10801, Springer International Publishing, Cham, 2018, pp. 711–738.

[8] McIver, A. and Morgan, C.: Partial Correctness for Probabilistic Demonic Programs, *Theoretical Computer Science*, Vol. 266, No. 1-2(2001), pp. 513–541.