Staged Composition Synthesis

Boris Düdder Moritz Martens Jakob Rehof

boris.duedder@cs.tu-dortmund.de
moritz.martens@cs.tu-dortmund.de
jakob.rehof@cs.tu-dortmund.de
Fakultät für Informatik
Technischen Universität Dortmund 44227 Dortmund

Abstract: A framework for composition synthesis is provided in which metalanguage combinators are supported and the execution of synthesized programs can be staged into composition-time code generation (stage 1) and run-time execution (stage 2). By extending composition synthesis to encompass both object language (L1) and metalanguage (L2) combinators, composition synthesis becomes a powerful and flexible framework for the generation of L1-program compositions. A system of modal intersection types is introduced into a combinatory composition language to control the distinction between L1- and L2-combinators at the type level, thereby exposing the language distinction to composition synthesis. We provide a theory of correctness of the framework which ensures that generated compositions of component implementations are well typed and that their execution can be staged such that all metalanguage combinators can be computed away completely at stage 1, leaving only well typed L1-code for execution at stage 2. Furthermore, we report on experiments.

Composition synthesis [Reh13] is based on the idea of using inhabitation in combinatory logic with intersection types [BCDC83] as a foundation for computing compositions from a repository of components. We can regard a combinatory type judgement $\Gamma \vdash e : \tau$ as modeling the fact that combinatory expression e can be obtained by composition from a repository Γ of components which are exposed as combinator symbols and whose interfaces are exposed as combinator types enriched with intersection types that specify semantic properties of components. The decision problem of inhabitation, often indicated as $\Gamma \vdash ? : \tau$, is the question whether a combinatory expression e exists such that $\Gamma \vdash e : \tau$ (such an expression e is called an inhabitant of τ). An algorithm (or semi-algorithm) for solving the inhabitation problem searches for inhabitants and can be used to synthesize them. Under the propositions-as-types correspondence, inhabitation is the question of provability in a Hilbert-style presentation of a propositional logic, where Γ represents a propositional theory, τ represents a proposition to be proved, and e is a proof.

Following [WY05], a level of *semantic types* is introduced to specify component interfaces and synthesis goals so as to direct synthesis by means of semantic concepts. Semantic types are not necessarily checked against component implementations (this is regarded as an orthogonal issue). In combinatory logic synthesis (CLS) [DMRU12, Reh13, DMR13] semantic types are represented by intersection types [BCDC83]. In addition to being inherently component-oriented, it is a possible advantage of the type-based approach of

composition synthesis that types can be naturally associated with code at the API-level. We think of intersection types as hosting a two-level type system, consisting of *native types* and *semantic types*. Native types are types of the implementation language, whereas semantic types are abstract, application-dependent conceptual structures, drawn, e.g., from a taxonomy of semantic concepts.

In order to flexibilize CLS, staged composition synthesis (SCS) was proposed in [DMR14]. SCS introduces a metalanguage, L2, in which native implementation code, L1, (e.g. Java or ML) can be manipulated (e.g., complex L1-code substitutions are possible). The metalanguage is essentially the $\lambda_e^{\Box\rightarrow}$-calculus of Davies and Pfenning [DP01] which introduces a modal type operator, $\Box$, to inject L1-types into the type-language of L2. Intuitively, a type $\Box\tau$ can be understood to describe L1-code of L1-type τ or, informally speaking, $\Box\tau$ means "code of L1-type τ". A second repository containing *composition components* with implementations in L2 is introduced. Then, synthesis automatically composes both L1- and L2-components, resulting in more flexible and powerful forms of composition since complex (and usually context-specific) L1-code-maniualations, including substitutions of code into L1-templates, may be encapsulated in composition components. It is a nice consequence of the operational semantic theory of $\lambda_e^{\Box\rightarrow}$ that computation can be *staged*. For a composition e of type $\Box\tau$, it is guaranteed that all L2-operations can be computed away in a first *composition time* stage, leaving a well typed L1-program of type τ to be executed in a following *runtime* stage.

Our framework has been implemented in an extension of the (CL)S (Combinatory Logic Synthesizer) tool, and we report on the results of experiments using the tool for SCS.

References

[BCDC83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

[DMR13] Boris Düdder, Moritz Martens, and Jakob Rehof. Intersection Type Matching with Subtyping. In *Proceedings of TLCA'13*, volume 7941 of *LNCS*. Springer, 2013.

[DMR14] Boris Düdder, Moritz Martens, and Jakob Rehof. Staged Composition Synthesis. In *Proceedings of ESOP'14*, volume 8410 of *LNCS*, pages 67–86. Springer, 2014.

[DMRU12] Boris Düdder, Moritz Martens, Jakob Rehof, and Paweł Urzyczyn. Bounded Combinatory Logic. In *Proceedings of CSL'12*, volume 16 of *LIPIcs*, pages 243–258. Schloss Dagstuhl, 2012.

[DP01] Rowan Davies and Frank Pfenning. A Modal Analysis of Staged Computation. *Journal of the ACM*, 48(3):555–604, 2001.

[Reh13] Jakob Rehof. Towards Combinatory Logic Synthesis. In *BEAT'13, 1st International Workshop on Behavioural Types*. ACM, January 22 2013.

[WY05] Joe B. Wells and Boris Yakobowski. Graph-Based Proof Counting and Enumeration with Applications for Program Fragment Synthesis. In *LOPSTR 2004*, volume 3573 of *LNCS*, pages 262–277. Springer, 2005.