Compilation as Multi-Language Semantics

William J. Bowman
University of British Columbia
Vancouver, BC, CA
wjb@williamjbowman.com

ACM Reference Format:
William J. Bowman. 2021. Compilation as Multi-Language Semantics. In *Workshop on Principles of Secure Compilation.* ACM, New York, NY, USA, 3 pages.

$$\begin{aligned} A.v &::= .... \mid (\text{AS } S.e) \\ A.n &::= .... \mid (\text{AS } S.e) \\ A.e &::= .... \mid (\text{AS } S.e) \end{aligned} \qquad \begin{aligned} S.e &::= .... \mid (\text{SA } A.e) \\ e &::= S.e \mid A.e \end{aligned}$$

Fig. 1. λ^{sa} Syntax (excerpts)

1 Extended Abstract

Modeling interoperability between programs in different languages is a key problem when modeling compositional and secure compilation. Multi-language semantics provide a syntactic method for modeling language interopability (Matthews 2007), and has proven useful in compiler correctness and secure compilation (Ahmed 2015; Ahmed and Blume 2011; New et al. 2016; Patterson and Ahmed 2017; Perconti and Ahmed 2014).

Unfortunately, existing models of compilation using multi-language semantics duplicate effort. Two variants of each compiler pass are defined: a syntactic translation on open terms, and a run-time translation of closed terms at multi-language boundaries (Ahmed and Blume 2011; New et al. 2016). One must then prove that both definitions coincide.

We introduce a novel work-in-progress approach to uniformly model both variants as a single reduction system on open terms in a multi-language semantics. This simultaneously defines the compiler and the interoperability semantics. It also has interesting semantic consequences: different reduction strategies model different compilation strategies, and standard theorems about reduction imply standard compiler correctness theorems. For example, we get a model of ahead-of-time (AOT) compilation by normalizing cross-language redexes; the normal form with respect to these redexes is a target language term. We model just-in-time (JIT) compilation as nondeterministic evaluation in the multi-language: a term can either step in the source, or translate then step in the target. We prove that confluence of multi-language reduction implies compiler correctness and part of full abstraction; and that subject reduction implies type-preservation of the compiler.

An example instance: Reduction to A-normal form Our approach generalizes from high-level to low-level transformations of a wide array of language features. To demonstrate this, we have developed a 5-pass model compiler from a Scheme-like language to an x86-64-like language. Here, we model one interesting compiler pass: reduction to A-normal form (ANF). This pass is a good example and stress test. The A-reductions are tricky to define because they reorder a term with respect to its context, while the other passes locally transform a term in an arbitrary context.

The source is a standard dynamically typed functional imperative language, modeled on Scheme. It has a call-by-value heap-based small-step semantics, $(H\ S.e_1) \xrightarrow{\lambda s} (H\ S.e_2)$, where H represents the heap and $S.e$ is represents a source expression.[1] We omit the syntax and reduction rules for brevity.

The target language is essentially the same, but the syntax is restricted to A-normal form: all computations $A.n$ require values $A.v$ as operands; expressions $A.e$ cannot be nested and only explicitly compose and sequence intermediate computations $A.n$. The reduction relation, $(H\ A.e_1) \xrightarrow{\lambda a} (H\ A.e_2)$, does not require a control stack.

To develop a multi-language semantics, we embed syntactic terms from each language into a single syntax, defined in Figure 1. We extend each meta-variable with boundary terms (SA $A.e$) ("Source on the outside, ANF on the inside") and (AS $S.e$) ("ANF on the outside, Source on the inside").

The translation to ANF can be viewed as a reduction system in the multi-language. We define the A-reductions in Figure 2. These rules are essentially standard (Flanagan et al. 1993), but we modify them to make boundary transitions explicit. The A-reductions have the form $S.e \longrightarrow^{\text{a}} S.e$, reducing source expressions in the multi-language. Each A-reduction rewrites a source expression in a source evaluation context, transforming the control stack into a data stack. For example, the A-lift rule lifts a trivial computation, let-binding it and providing the let-bound name (a value) in evaluation position, explicitly sequencing the computation $A.n$ with the evaluation context $S.E$. The side-conditions syntactically encode termination conditions, preventing A-reductions of target redexes and in empty evaluation contexts.

[1] We use a prefix followed by a dot (.) to distinguish terms in each language—the prefix S for source terms and the prefix A for ANF terms.

$$A.e \longrightarrow^{\text{a}} (\text{SA } A.e) \qquad \text{[A-normal]}$$

$$\begin{aligned}&S.E[(\text{let } ([A.x\ S.e] \dots)\ S.e_2)] \qquad \text{[A-merge-l]}\\ \longrightarrow^{\text{a}}\ &(\text{SA } (\text{let } ([A.x\ (\text{AS } S.e)] \dots)\ (\text{AS } S.E[S.e_2])))\end{aligned}$$

$$\begin{aligned}&S.E[(\text{begin } S.e_r \dots\ S.e)] \qquad \text{[A-merge-b]}\\ \longrightarrow^{\text{a}}\ &(\text{SA } (\text{begin } (\text{AS } S.e_r) \dots (\text{AS } S.E[S.e])))\end{aligned}$$

$$\begin{aligned}&S.E[(\text{if } A.v\ S.e_1\ S.e_2)] \qquad \text{[A-join]}\\ \longrightarrow^{\text{a}}\ &(\text{SA } (\text{letrec } ([\text{j } (\lambda\ (x)\ S.E[x])])\\ &\qquad (\text{if } A.v\ (\text{AS } (\text{j } S.e_1))\ (\text{AS } (\text{j } S.e_2)))))\\ &\text{where } S.E \notin A.Cn,\ \text{j},\ x \text{ fresh}\end{aligned}$$

$$\begin{aligned}&S.E[A.n] \qquad \text{[A-lift]}\\ \longrightarrow^{\text{a}}\ &(\text{SA } (\text{let } ([x\ A.n])\ (\text{AS } S.E[x])))\\ &\text{where } S.E \notin A.Cn,\ A.n \notin A.v,\ x \text{ fresh}\end{aligned}$$

Fig. 2. The A-reductions (excerpts)

$$C[(\text{AS } (\text{SA } e))] \longrightarrow^{\text{st}} C[e]$$

$$C[(\text{SA } (\text{AS } e))] \longrightarrow^{\text{st}} C[e]$$

Fig. 3. λ^{sa} Boundary Reductions

$$T ::= C[(\text{AS } A.Cm)] \qquad \frac{e_1 \longrightarrow^{\text{a}} e}{T[e_1] \ {}^{\text{s}}\!\!\longrightarrow^{\text{a}} T[e]} \qquad \frac{e_1 \longrightarrow^{\text{st}} e}{e_1 \ {}^{\text{s}}\!\!\longrightarrow^{\text{a}} e}$$

Fig. 4. λ^{sa} Translation Reductions

We supplement the multi-language A-reductions with the standard boundary cancellation reductions, given in Figure 3. These apply under any multi-language context C.

In Figure 4 we define the translation reductions. These extend the A-reductions to apply under any translation context T. The construction of the translation context for ANF is a little unusual, but the intuition is simple: a translation context identifies a pure source expression under any context, including under a target/source boundary. The context $A.Cm$ corresponds to an ANF context that can have any expression in the hole. In one step, the translation reductions can perform either one A-reduction or one boundary cancellation.

From the translation reductions, we derive AOT compilation as normalization with respect to translation reductions.

Definition 1 (ANF Compilation by Normalization).

$$\frac{(\text{AS } S.e) \ {}^{\text{s}}\!\!\longrightarrow^{\text{a}*} A.e \qquad A.e \ {}^{\text{s}}\!\!\not\longrightarrow^{\text{a}}}{S.e \Downarrow^{\text{anf}} A.e}$$

Finally, we define the multi-language semantics in Figure 5. This defines all possible transitions in the multi-language. A term can either take a step in the source language, or a translation step, or a step in the target language. Multi-language reduction is indexed by a heap, H, which is used by the source and target reductions but not the translation reductions.

$$\frac{(H_1\ S.e_1) \xrightarrow{\lambda\text{s}} (H_2\ S.e_2)}{(H_1\ S.e_1) \overset{\lambda\text{sa}}{\Longrightarrow} (H_2\ S.e_2)} \qquad \frac{(H_1\ S.e_1) \xrightarrow{\lambda\text{s}} (H_2\ S.e_2)}{(H_1\ (\text{AS } S.e_1)) \overset{\lambda\text{sa}}{\Longrightarrow} (H_2\ (\text{AS } S.e_2))}$$

$$\frac{(H_1\ A.e_1) \xrightarrow{\lambda\text{a}} (H_2\ A.e_2)}{(H_1\ A.e_1) \overset{\lambda\text{sa}}{\Longrightarrow} (H_2\ A.e_2)} \qquad \frac{(H_1\ A.e_1) \xrightarrow{\lambda\text{a}} (H_2\ A.e_2)}{(H_1\ (\text{SA } A.e_1)) \overset{\lambda\text{sa}}{\Longrightarrow} (H_2\ (\text{SA } A.e_2))}$$

$$\frac{A.e_1 \ {}^{\text{s}}\!\!\longrightarrow^{\text{a}} A.e_2}{(H_1\ A.e_1) \overset{\lambda\text{sa}}{\Longrightarrow} (H_1\ A.e_2)}$$

Fig. 5. λ^{sa} Multi-language Reduction

Note that terms already in the heap are not translated, which corresponds to an assumption that the language memory models are identical. We could lift this restriction by adding multi-language boundaries to heap values and extending translation reductions to apply in the heap.

The multi-language reduction allows reducing in the source, modeling interpretation, or translating then reducing in the target, modeling JIT compilation before continuing execution. This does not model speculative optimization; equipping the multi-language with assumption instructions as done by Flückiger et al. (2018) might support modeling this.

Standard meta-theoretic properites of reduction impliy standard compiler correctness results.

Subject reduction of the multi-language semantics implies type-preservation of the compiler. This is simple for our present compiler, since the type system is simple, but the theorems applies for more complex type systems.

Theorem 1 (Subject Reduction implies Type Preservation). *If* $(\Gamma \vdash e_1 : \tau$ *and* $e_1 \ {}^{\text{s}}\!\!\longrightarrow^{\text{a}*} e_2$ *implies* $\Gamma \vdash e_2 : \tau)$ *then* $(S.\Gamma \vdash S.e : S.\tau$ *and* $S.e \Downarrow^{\text{anf}} A.e$ *implies* $\exists A.\Gamma, A.\tau.\ A.\Gamma \vdash A.e : A.\tau)$.

We derive compiler correctness from confluence.

Conjecture 1 (Confluence). *If* $(H\ e) \overset{\lambda\text{sa}}{\Longrightarrow}{}^{*} (H_1\ e_1)$ *and* $(H\ e) \overset{\lambda\text{sa}}{\Longrightarrow}{}^{*} (H_2\ e_2)$ *then* $(H_1\ e_1) \overset{\lambda\text{sa}}{\Longrightarrow}{}^{*} (H_3\ e_3)$ *and* $(H_2\ e_2) \overset{\lambda\text{sa}}{\Longrightarrow}{}^{*} (H_3\ e_3)$

Note the multi-language semantics can reduce open terms, so confluence implies correctness of both the AOT and the JIT compiler. As an example, whole-program correctness is a trivial corollary of confluence.

Corollary 2 (Whole-Program Correctness). *If* $(()\ S.e) \xrightarrow{\lambda\text{s}}{}^{*} (H\ S.v)$ *and* $S.e \Downarrow^{\text{anf}} A.e$ *then* $(()\ A.e) \xrightarrow{\lambda\text{a}}{}^{*} (H\ A.v)$ *such that* $A.v$ *is equal to* $S.v$.

Multi-language semantics provide a strong attacker model through contextual equivalence. A context C models an attacker that can provide either source or target code or data as input and observe the result. Contextual equivalence is extended to relate reduction configurations, not just terms, to enable the definition to apply to the JIT model.

Definition 2 (Contextual Equivalence). *$(H_1\ e_1) \approx (H_2\ e_2)$ if for all multi-language contexts C, $(H_1\ C[e_1])$ and $(H_2\ C[e_2])$ co-terminate in $\overset{\lambda sa}{\Rightarrow}$.*

We define secure compilation of both the AOT and JIT models as full abstraction: contextual equivalence is preserved and reflected through multi-language reduction.

Theorem 3 (Full Abstraction (multi-language)). *Suppose $(H_1\ e_1) \overset{\lambda sa}{\Longrightarrow} (H_1'\ e_1')$ and $(H_2\ e_2) \overset{\lambda sa}{\Longrightarrow} (H_2'\ e_2')$. Then $(H_1\ e_1) \approx (H_2\ e_2)$ if and only if $(H_1'\ e_1') \approx (H_2'\ e_2')$.*

The normally easy part of full abstraction, within the multi-language, is now a direct consequence of confluence, since both compilation and contextual equivalence are defined by multi-language reduction. The hard part, showing any multi-language context (attacker) is emulated by a source context, remains.

Bibliography

Amal Ahmed. Verified Compilers for a Multi-language World. In *Proc. Summit oN Advances in Programming Languages (SNAPL)*, 2015. doi:10.4230/LIPIcs.SNAPL.2015.15

Amal Ahmed and Matthias Blume. An Equivalence-Preserving CPSTranslation via Multi-Language Semantics. In *Proc. International Conference on Functional Programming (ICFP)*, 2011. doi:10.1145/2034773.2034830

Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The Essence of Compiling with Continuations. In *Proc. International Conference on Programming Language Design and Implementation (PLDI)*, 1993. doi:10.1145/155090.155113

Olivier Flückiger, Gabriel Scherer, Ming-Ho Yee, Aviral Goel, Amal Ahmed, and Jan Vitek. Correctness of speculative optimizations with dynamic deoptimization. *Proceedings of the ACMon Programming Languages (PACMPL)* 2(POPL), pp. 1–28, 2018. doi:10.1145/3158137

Robert Bruce Matthews Jacob And Findler. Operational Semantics for Multi-language Programs. In *Proc. Symposium on Principles of Programming Languages (POPL)*, 2007. doi:10.1145/1190216.1190220

Max S. New, William J. Bowman, and Amal Ahmed. Fully Abstract Compilation via Universal Embedding. In *Proc. International Conference on Functional Programming (ICFP)*, 2016. doi:10.1145/2951913.2951941

Daniel Patterson and Amal Ahmed. Linking Types for Multi-Language Software: Have Your Cake and Eat It Too. In *Proc. Summit oN Advances in Programming Languages (SNAPL)*, 2017. doi:10.4230/LIPIcs.SNAPL.2017.12

James T. Perconti and Amal Ahmed. Verifying an Open Compiler Using Multi-language Semantics. In *Proc. European Symposium on Programming (ESOP)*, 2014. doi:10.1007/978-3-642-54833-8_8