15-791 Project
Compiling by Push-Value

Zach Battleman (zbattlem)
Sonya Simkin (ssimkin)

1 Our Project So Far

Here is a summary of the progress we have made thus far with our project:

- We have set up the general infrastructure for the compiler in OCaml, including code to accept source files in our ML-subset language as command-line arguments for compilation
- We have defined the grammar for our ML-subset language, and we have implemented the lexing and parsing of source files into an AST
- We have defined the ASTs for both the source language and for the CBPV IR
- We have implemented typechecking for the source language AST
- We have implemented the translation from the source language to CBPV

We have finished the programming portion of our safety goal, and the written component should flow smoothly now that we have the translation done, albeit in Ocaml. Admittedly, we have not written that much because getting off the ground in Ocaml was a bit trickier than we anticipated, but writing up the proofs for our safety goal should hopefully not be too difficult, and we expect implementing our regular goals should not be too difficult.

2 Introduction

TBD

3 Our Compiler

3.1 Source Language

The source language for our compiler is a limited ML-like language that contains functions, sums, positive products, and a print effect. The grammar for the language is given in Figure 1 (note that the s in the `print` construct is a string constant).

3.2 Compiler Pipeline

3.2.1 ML to CBPV

The first step of our translation is embedding our ML fragment into CBPV. This is a fairly straightforward process taking inspiration from (TODO cite). The main departure is we don't have an

Typ	τ	::=	`unit`	unit	`unit`
			`prod`$(\tau_1;\tau_2)$	$\tau_1 \times \tau_2$	`tau1 * tau2`
			`sum`$(\tau_1;\tau_2)$	$\tau_1 + \tau_2$	`tau1 + tau2`
			`arr`$(\tau_1;\tau_2)$	$\tau_1 \to \tau_2$	`tau1 -> tau2`
Term	m	::=	x	x	`x`
			`triv`	$\langle\rangle$	`<>`
			`check`$(m;m')$	`check` m `in` m'	`check m in m'`
			`pair`$(m_1;m_2)$	$\langle m_1, m_2\rangle$	`<m1, m2>`
			`split`$(m;x_1.x_2.m')$	`split` m `as` x_1, x_2 `in` m'	`split m as x1, x2 in m'`
			`inj`$[\text{sum}(\tau_1;\tau_2)](1)(m)$	`inj`$[\tau_1 + \tau_2](1)(m)$	`inj[tau1 + tau2](1)(m)`
			`inj`$[\text{sum}(\tau_1;\tau_2)](2)(m)$	`inj`$[\tau_1 + \tau_2](2)(m)$	`inj[tau1 + tau2](2)(m)`
			`case`$(m;x_1.m_1;x_2.m_2)$	`case` $m\{x_1.m_1 \mid x_2.m_2\}$	`case e {x1 => e1 | x2 => e2}`
			$\lambda(x:\tau.m)$	$\lambda(x:\tau.m)$	`fn (x : tau) => m`
			`ap`$(m;m_1)$	$m(m_1)$	`e e1`
			`print`$(s;m)$	`print` s; m	`print s; m`

Figure 1: Source language grammar

explicit Lax IR. (Justify why) We translate types in the following manner:

$$\begin{aligned}
\|\mathsf{Unit}\| &\rightsquigarrow \mathsf{Unit} \\
\|A_1 \to A_2\| &\rightsquigarrow U(\|A_1\| \to F(\|A_2\|)) \\
\|A_1 \times A_2\| &\rightsquigarrow \|A_1\| \otimes \|A_2\| \\
\|A_1 + A_2\| &\rightsquigarrow \|A_1\| + \|A_2\|
\end{aligned}$$

Additionally, we translate terms as (todo)

With this translation, we arrive at the following theorem:

Theorem 1. *If* $\Gamma \vdash_{\mathsf{ML}} e : A$ *then* $\|\Gamma\| \vdash_{\mathsf{CPBV}} \|e\| : F(\|A\|)$

Proof. Todo. Induction □

Observe how this embedding into CBPV codifies the intuition of ML. Consider the ML term $(\langle\rangle, \mathsf{Print}\ \text{“hello”}; \langle\rangle)$. This is translated to CBPV term `Bind(Ret(()); var_1.Bind(Bind(Print("meow"); var_3.Ret(())); var_2.Ret((var_1,var_2))))` (make this latex). This precisely captures the notion of evaluating the first component, evaluating the second component, and then giving back the true, value tuple.

4 Conclusion

TBD

References

[1] Robert Harper. *Call-by-Push-Value and the Enriched Effects Calculus*. 2024. URL: https://www.cs.cmu.edu/~rwh/courses/atpl/pdfs/cbpv.pdf.

[2] Max S. New. *Compiling with Call-by-push-value*. 2023. URL: https://maxsnew.com/docs/mfps2023-slides.pdf.