# Towards verified compilation of CakeML into WebAssembly

Lorenz Leutgeb
`lorenz@leutgeb.xyz`

2018-09-19

TU Wien and Data61, CSIRO

# How did I get here?

- Previously: Bachelor's in Software Engineering at TU Wien
- Currently: European Master's Program in Computational Logic
  - TU Dresden, Germany, EU
  - Free University of Bolzano, Italy, EU
  - TU Wien, Austria, EU
  - Ekaterina Lebedeva is an alumna!
  - Rajeev Gore will be at TU Wien next week!
  - Peter Baumgartner matched me with Michael Norrish!
- My project work is sponsored in part by EMCL and by Data61, and I am supervised by Michael Norrish.

# Agenda

Introduction

- WebAssembly
- CakeML

Translating CakeML to WebAssembly

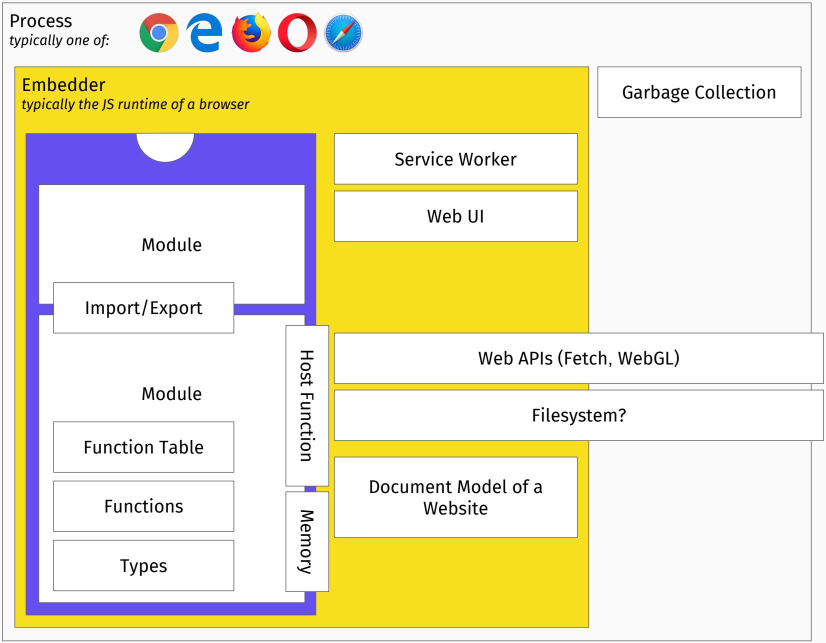
Verification

Progress and Outlook

# Introduction

WA

WEBASSEMBLY


Process
typically one of:
Embedder
typically the JS runtime of a browser
Garbage Collection
Service Worker
Web UI
Module
Import/Export
Module
Function Table
Functions
Types
Host Function
Memory
Web APIs (Fetch, WebGL)
Filesystem?
Document Model of a Website

# What is WebAssembly?...

"Efficient, safe, sandboxed, universal, open, public, formal, verified, clean, simple."

- A portable code format and instruction set architecture
- Sandboxed, embedded by design
- Open Standard, steered by a World Wide Web Consortium working group
- Actually supported by all major browser vendors
- Supersedes PNaCl (*portable native client*, Google), asm.js (simple subset of JS, Mozilla)
- Within 10% of native code performance in the browser
- Open up browsers for other languages than JS

# Design Goals

| Semantics | Representation |
| --- | --- |
| Language independent | Compact |
| Platform independent | Easy to generate |
| Hardware independent | Fast to decode |
| Fast to execute | Fast to validate |
| Safe to execute | Fast to compile |
| Deterministic | Streamable |
| Easy to reason about | Parallelisable |

Taken from presentation “Neither Web nor Assembly” by Andreas Rossberg (spec author).

# Concrete Syntax by Example

S-Expression:

```
(module
  (func $add
    (param i32 i32)
    (result i32)
    (i32.add         ;; 3
      (get_local 0)  ;; 1
      (get_local 1)  ;; 2
    )
  )
  (export ”add” (func $add))
)
```

```
00 6100 6d73 0001 0000 0701 6001 7f02 017f
10 037f 0102 0700 010a 6106 6464 7754 006f
20 0a00 0109 0007 0020 0120 0b6a 1900 6e04
30 6d61 0165 0109 0600 6461 5464 6f77 0702
40 0001 0002 0100 0000
```

Preferred by browsers, for bandwidth!

# Concrete Syntax by Example

S-Expression:

```
(module
  (func $add
    (param i32 i32)
    (result i32)
    (i32.add          ;; 3
      (get_local 0)   ;; 1
      (get_local 1)   ;; 2
    )
  )
  (export "add" (func $add))
)
```

```
00 6100 6d73 0001 0000 0701 6001 7f02 017f
10 037f 0102 0700 010a 6106 6464 7754 006f
20 0a00 0109 0007 0020 0120 0b6a 1900 6e04
30 6d61 0165 0109 0600 6461 5464 6f77 0702
40 0001 0002 0100 0000
```

Preferred by browsers, for bandwidth!

Postfix notation:

```
(module
  (func $fac
    (param f64) (result f64)
    (get_local 0)
    (f64.const 1)
    f64.lt
    if (result f64)
      f64.const 1
    else
      get_local 0
      get_local 0
      f64.const 1
      f64.sub
      call $fac
      f64.mul
    end
  )
)
```

# Abstract Syntax

- Instructions

  **Numeric** Constants, unary and binary operations, binary relations, conversions.

  **Variables** `get_local`, `set_local`, `get_global`, `set_global`

  **Memory** `load`, `store`, `size`, `grow`

  **Control Flow** `if`, `loop`, `block`, `call`. All of them have some notion of a return type.

  **...** and a few others.

- Declarations for tables, memories, exports, imports, ...

$$
\begin{array}{lrcl}
\text{(value types)} & t & ::= & \mathsf{i32} \mid \mathsf{i64} \mid \mathsf{f32} \mid \mathsf{f64} \\
\text{(packed types)} & tp & ::= & \mathsf{i8} \mid \mathsf{i16} \mid \mathsf{i32} \\
\text{(function types)} & tf & ::= & t^* \to t^* \\
\text{(global types)} & tg & ::= & \mathsf{mut}^? \; t
\end{array}
$$

$$
\begin{array}{rcl}
unop_{\mathrm{i}N} & ::= & \mathbf{clz} \mid \mathbf{ctz} \mid \mathbf{popcnt} \\
unop_{\mathrm{f}N} & ::= & \mathbf{neg} \mid \mathbf{abs} \mid \mathbf{ceil} \mid \mathbf{floor} \mid \mathbf{trunc} \mid \mathbf{nearest} \mid \mathbf{sqrt} \\
binop_{\mathrm{i}N} & ::= & \mathbf{add} \mid \mathbf{sub} \mid \mathbf{mul} \mid \mathbf{div}\_sx \mid \mathbf{rem}\_sx \mid \\
 & & \mathbf{and} \mid \mathbf{or} \mid \mathbf{xor} \mid \mathbf{shl} \mid \mathbf{shr}\_sx \mid \mathbf{rotl} \mid \mathbf{rotr} \\
binop_{\mathrm{f}N} & ::= & \mathbf{add} \mid \mathbf{sub} \mid \mathbf{mul} \mid \mathbf{div} \mid \mathbf{min} \mid \mathbf{max} \mid \mathbf{copysign} \\
testop_{\mathrm{i}N} & ::= & \mathbf{eqz} \\
relop_{\mathrm{i}N} & ::= & \mathbf{eq} \mid \mathbf{ne} \mid \mathbf{lt}\_sx \mid \mathbf{gt}\_sx \mid \mathbf{le}\_sx \mid \mathbf{ge}\_sx \\
relop_{\mathrm{f}N} & ::= & \mathbf{eq} \mid \mathbf{ne} \mid \mathbf{lt} \mid \mathbf{gt} \mid \mathbf{le} \mid \mathbf{ge} \\
cvtop & ::= & \mathbf{convert} \mid \mathbf{reinterpret} \\
sx & ::= & \mathbf{s} \mid \mathbf{u}
\end{array}
$$

$$
\begin{array}{lrcl}
\text{(instructions)} & e & ::= & \mathbf{unreachable} \mid \mathbf{nop} \mid \mathbf{drop} \mid \mathbf{select} \mid \\
 & & & \mathbf{block}\; tf\; e^*\; \mathbf{end} \mid \mathbf{loop}\; tf\; e^*\; \mathbf{end} \mid \mathbf{if}\; tf\; e^*\; \mathbf{else}\; e^*\; \mathbf{end} \mid \\
 & & & \mathbf{br}\; i \mid \mathbf{br\_if}\; i \mid \mathbf{br\_table}\; i^+ \mid \mathbf{return} \mid \mathbf{call}\; i \mid \mathbf{call\_indirect}\; tf \mid \\
 & & & \mathbf{get\_local}\; i \mid \mathbf{set\_local}\; i \mid \mathbf{tee\_local}\; i \mid \mathbf{get\_global}\; i \mid \\
 & & & \mathbf{set\_global}\; i \mid t.\mathbf{load}\; (tp\_sx)^?\; a\; o \mid t.\mathbf{store}\; tp^?\; a\; o \mid \\
 & & & \mathbf{current\_memory} \mid \mathbf{grow\_memory} \mid t.\mathbf{const}\; c \mid \\
 & & & t.unop_t \mid t.binop_t \mid t.testop_t \mid t.relop_t \mid t.cvtop\; t\_sx^?
\end{array}
$$

$$
\begin{array}{lrcl}
\text{(functions)} & f & ::= & ex^*\; \mathbf{func}\; tf\; \mathbf{local}\; t^*\; e^* \mid ex^*\; \mathbf{func}\; tf\; im \\
\text{(globals)} & glob & ::= & ex^*\; \mathbf{global}\; tg\; e^* \mid ex^*\; \mathbf{global}\; tg\; im \\
\text{(tables)} & tab & ::= & ex^*\; \mathbf{table}\; n\; i^* \mid ex^*\; \mathbf{table}\; n\; im \\
\text{(memories)} & mem & ::= & ex^*\; \mathbf{memory}\; n \mid ex^*\; \mathbf{memory}\; n\; im \\
\text{(imports)} & im & ::= & \mathbf{import}\; \text{“}name\text{”}\; \text{“}name\text{”} \\
\text{(exports)} & ex & ::= & \mathbf{export}\; \text{“}name\text{”} \\
\text{(modules)} & m & ::= & \mathbf{module}\; f^*\; glob^*\; tab^?\; mem^?
\end{array}
$$

**Figure 1.** WebAssembly abstract syntax

$$
\begin{array}{llcll}
\text{(store)} & s & ::= & \{\mathsf{inst}\ \mathit{inst}^*,\ \mathsf{tab}\ \mathit{tabinst}^*,\ \mathsf{mem}\ \mathit{meminst}^*\} & \\
\text{(instances)} & \mathit{inst} & ::= & \{\mathsf{func}\ \mathit{cl}^*,\ \mathsf{glob}\ v^*,\ \mathsf{tab}\ i^?,\ \mathsf{mem}\ i^?\} & \\
 & \mathit{tabinst} & ::= & \mathit{cl}^* & \\
 & \mathit{meminst} & ::= & b^* & \\
\text{(closures)} & \mathit{cl} & ::= & \{\mathsf{inst}\ i,\ \mathsf{code}\ f\} & \text{(where } f \text{ is not an import and has all exports } \mathit{ex}^* \text{ erased)} \\
\text{(values)} & v & ::= & t.\mathbf{const}\ c & \\
\text{(administrative operators)} & e & ::= & \dots \mid \mathbf{trap} \mid \mathbf{call}\ \mathit{cl} \mid \mathbf{label}_n\{e^*\}\ e^*\ \mathbf{end} \mid \mathbf{local}_n\{i; v^*\}\ e^*\ \mathbf{end} & \\
\text{(local contexts)} & L^0 & ::= & v^*\ [\_]\ e^* & \\
 & L^{k+1} & ::= & v^*\ \mathbf{label}_n\{e^*\}\ L^k\ \mathbf{end}\ e^* &
\end{array}
$$

**Reduction**

$$
\frac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v^*; L^k[e^*] \hookrightarrow_i s'; v'^*; L^k[e'^*]}
\qquad
\frac{s; v^*; e^* \hookrightarrow_i s'; v'^*; e'^*}{s; v_0^*; \mathbf{local}_n\{i; v^*\}\ e^*\ \mathbf{end} \hookrightarrow_j s'; v_0^*; \mathbf{local}_n\{i; v'^*\}\ e'^*\ \mathbf{end}}
\qquad
\boxed{s; v^*; e^* \hookrightarrow_i s; v^*; e^*}
$$

$$
\begin{array}{rcll}
L^0[\mathbf{trap}] & \hookrightarrow & \mathbf{trap} & \text{if } L^0 \neq [\_] \\[4pt]
(t.\mathbf{const}\ c)\ t.\mathit{unop} & \hookrightarrow & t.\mathbf{const}\ \mathit{unop}_t(c) & \\
(t.\mathbf{const}\ c_1)\ (t.\mathbf{const}\ c_2)\ t.\mathit{binop} & \hookrightarrow & t.\mathbf{const}\ c & \text{if } c = \mathit{binop}_t(c_1, c_2) \\
(t.\mathbf{const}\ c_1)\ (t.\mathbf{const}\ c_2)\ t.\mathit{binop} & \hookrightarrow & \mathbf{trap} & \text{otherwise} \\
(t.\mathbf{const}\ c)\ t.\mathit{testop} & \hookrightarrow & \mathbf{i32.const}\ \mathit{testop}_t(c) & \\
(t.\mathbf{const}\ c_1)\ (t.\mathbf{const}\ c_2)\ t.\mathit{relop} & \hookrightarrow & \mathbf{i32.const}\ \mathit{relop}_t(c_1, c_2) & \\[4pt]
(t_1.\mathbf{const}\ c)\ t_2.\mathbf{convert}\ t_1\_\mathit{sx}^? & \hookrightarrow & t_2.\mathbf{const}\ c' & \text{if } c' = \mathrm{cvt}^{\mathit{sx}^?}_{t_1,t_2}(c) \\
(t_1.\mathbf{const}\ c)\ t_2.\mathbf{convert}\ t_1\_\mathit{sx}^? & \hookrightarrow & \mathbf{trap} & \text{otherwise} \\
(t_1.\mathbf{const}\ c)\ t_2.\mathbf{reinterpret}\ t_1 & \hookrightarrow & t_2.\mathbf{const}\ \mathrm{const}_{t_2}(\mathrm{bits}_{t_1}(c)) & \\[4pt]
\mathbf{unreachable} & \hookrightarrow & \mathbf{trap} & \\
\mathbf{nop} & \hookrightarrow & \epsilon & \\
v\ \mathbf{drop} & \hookrightarrow & \epsilon & \\
v_1\ v_2\ (\mathbf{i32.const}\ 0)\ \mathbf{select} & \hookrightarrow & v_2 & \\
v_1\ v_2\ (\mathbf{i32.const}\ k+1)\ \mathbf{select} & \hookrightarrow & v_1 & \\[4pt]
v^n\ \mathbf{block}\ (t_1^n \to t_2^m)\ e^*\ \mathbf{end} & \hookrightarrow & \mathbf{label}_m\{\epsilon\}\ v^n\ e^*\ \mathbf{end} & \\
v^n\ \mathbf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathbf{end} & \hookrightarrow & \mathbf{label}_n\{\mathbf{loop}\ (t_1^n \to t_2^m)\ e^*\ \mathbf{end}\}\ v^n\ e^*\ \mathbf{end} & \\
(\mathbf{i32.const}\ 0)\ \mathbf{if}\ \mathit{tf}\ e_1^*\ \mathbf{else}\ e_2^*\ \mathbf{end} & \hookrightarrow & \mathbf{block}\ \mathit{tf}\ e_2^*\ \mathbf{end} & \\
(\mathbf{i32.const}\ k+1)\ \mathbf{if}\ \mathit{tf}\ e_1^*\ \mathbf{else}\ e_2^*\ \mathbf{end} & \hookrightarrow & \mathbf{block}\ \mathit{tf}\ e_1^*\ \mathbf{end} & \\[4pt]
\mathbf{label}_n\{e^*\}\ v^*\ \mathbf{end} & \hookrightarrow & v^* & \\
\mathbf{label}_n\{e^*\}\ \mathbf{trap}\ \mathbf{end} & \hookrightarrow & \mathbf{trap} & \\
\mathbf{label}_n\{e^*\}\ L^j[v^n\ (\mathbf{br}\ j)]\ \mathbf{end} & \hookrightarrow & v^n\ e^* & \\
(\mathbf{i32.const}\ 0)\ (\mathbf{br\_if}\ j) & \hookrightarrow & \epsilon & \\
(\mathbf{i32.const}\ k+1)\ (\mathbf{br\_if}\ j) & \hookrightarrow & \mathbf{br}\ j & \\
(\mathbf{i32.const}\ k)\ (\mathbf{br\_table}\ j_1^k\ j\ j_2^*) & \hookrightarrow & \mathbf{br}\ j & \\
(\mathbf{i32.const}\ k+n)\ (\mathbf{br\_table}\ j_1^k\ j) & \hookrightarrow & \mathbf{br}\ j & \\[4pt]
s; \mathbf{call}\ j & \hookrightarrow_i & \mathbf{call}\ s_{\mathsf{func}}(i, j) & \\
s; (\mathbf{i32.const}\ j)\ \mathbf{call\_indirect}\ \mathit{tf} & \hookrightarrow_i & \mathbf{call}\ s_{\mathsf{tab}}(i, j) & \text{if } s_{\mathsf{tab}}(i, j)_{\mathsf{code}} = (\mathbf{func}\ \mathit{tf}\ \mathbf{local}\ t^*\ e^*) \\
s; (\mathbf{i32.const}\ j)\ \mathbf{call\_indirect}\ \mathit{tf} & \hookrightarrow_i & \mathbf{trap} & \text{otherwise} \\
v^n\ (\mathbf{call}\ \mathit{cl}) & \hookrightarrow & \mathbf{local}_m\{\mathit{cl}_{\mathsf{inst}}; v^n\ (t.\mathbf{const}\ 0)^k\}\ \mathbf{block}\ (\epsilon \to t_2^m)\ e^*\ \mathbf{end}\ \mathbf{end}\ \ \dots & \\
\mathbf{local}_n\{i; v_l^*\}\ v^n\ \mathbf{end} & \hookrightarrow & v^n & \mid\ \dots \text{if } \mathit{cl}_{\mathsf{code}} = (\mathbf{func}\ (t_1^n \to t_2^m)\ \mathbf{local}\ t^k\ e^*) \\
\mathbf{local}_n\{i; v_l^*\}\ \mathbf{trap}\ \mathbf{end} & \hookrightarrow & \mathbf{trap} & \\
\mathbf{local}_n\{i; v_l^*\}\ L^k[v^n\ \mathbf{return}]\ \mathbf{end} & \hookrightarrow & v^n & \\[4pt]
v_1^j\ v\ v_2^k; \mathbf{get\_local}\ j & \hookrightarrow & v & \\
v_1^j\ v\ v_2^k; v'\ (\mathbf{set\_local}\ j) & \hookrightarrow & v_1^j\ v'\ v_2^k; \epsilon & \\
v\ (\mathbf{tee\_local}\ j) & \hookrightarrow & v\ v\ (\mathbf{set\_local}\ j) & \\
s; \mathbf{get\_global}\ j & \hookrightarrow_i & s_{\mathsf{glob}}(i, j) & \\
s; v\ (\mathbf{set\_global}\ j) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \mathsf{glob}(i, j) = v \\[4pt]
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{load}\ a\ o) & \hookrightarrow_i & t.\mathbf{const}\ \mathrm{const}_t(b^*) & \text{if } s_{\mathsf{mem}}(i, k+o, |t|) = b^* \\
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{load}\ \mathit{tp}\_\mathit{sx}\ a\ o) & \hookrightarrow_i & t.\mathbf{const}\ \mathrm{const}_t^{\mathit{sx}}(b^*) & \text{if } s_{\mathsf{mem}}(i, k+o, |\mathit{tp}|) = b^* \\
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{load}\ \mathit{tp}\_\mathit{sx}^?\ a\ o) & \hookrightarrow_i & \mathbf{trap} & \text{otherwise} \\
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{const}\ c)\ (t.\mathbf{store}\ a\ o) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \mathsf{mem}(i, k+o, |t|) = \mathrm{bits}_t^{|t|}(c) \\
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{const}\ c)\ (t.\mathbf{store}\ \mathit{tp}\ a\ o) & \hookrightarrow_i & s'; \epsilon & \text{if } s' = s \text{ with } \mathsf{mem}(i, k+o, |\mathit{tp}|) = \mathrm{bits}_t^{|\mathit{tp}|}(c) \\
s; (\mathbf{i32.const}\ k)\ (t.\mathbf{const}\ c)\ (t.\mathbf{store}\ \mathit{tp}^?\ a\ o) & \hookrightarrow_i & \mathbf{trap} & \text{otherwise} \\
s; \mathbf{current\_memory}] & \hookrightarrow_i & \mathbf{i32.const}\ |s_{\mathsf{mem}}(i, *)|/64\,\mathrm{Ki} & \\
s; (\mathbf{i32.const}\ k)\ \mathbf{grow\_memory} & \hookrightarrow_i & s'; \mathbf{i32.const}\ |s_{\mathsf{mem}}(i, *)|/64\,\mathrm{Ki} & \text{if } s' = s \text{ with } \mathsf{mem}(i, *) = s_{\mathsf{mem}}(i, *)\ (0)^{k \cdot 64\,\mathrm{Ki}} \\
s; (\mathbf{i32.const}\ k)\ \mathbf{grow\_memory} & \hookrightarrow_i & \mathbf{i32.const}\ (-1) &
\end{array}
$$

**Figure 2.** Small-step reduction rules

(contexts) $C \;::=\; \{\mathsf{func}\ tf^*,\ \mathsf{global}\ tg^*,\ \mathsf{table}\ n^?,\ \mathsf{memory}\ n^?,\ \mathsf{local}\ t^*,\ \mathsf{label}\ (t^*)^*,\ \mathsf{return}\ (t^*)^?\}$

**Typing Instructions** $\boxed{C \vdash e^* : tf}$

$$\frac{}{C \vdash t.\mathbf{const}\ c : \epsilon \to t} \qquad \frac{}{C \vdash t.unop : t \to t} \qquad \frac{}{C \vdash t.binop : t\ t \to t} \qquad \frac{}{C \vdash t.testop : t \to \mathsf{i32}} \qquad \frac{}{C \vdash t.relop : t\ t \to \mathsf{i32}}$$

$$\frac{t_1 \neq t_2 \qquad sx^? = \epsilon \Leftrightarrow (t_1 = \mathsf{i}n \wedge t_2 = \mathsf{i}n' \wedge |t_1| < |t_2|) \vee (t_1 = \mathsf{f}n \wedge t_2 = \mathsf{f}n')}{C \vdash t_1.\mathbf{convert}\ t_2\_sx^? : t_2 \to t_1} \qquad \frac{t_1 \neq t_2 \qquad |t_1| = |t_2|}{C \vdash t_1.\mathbf{reinterpret}\ t_2 : t_2 \to t_1}$$

$$\frac{}{C \vdash \mathbf{unreachable} : t_1^* \to t_2^*} \qquad \frac{}{C \vdash \mathbf{nop} : \epsilon \to \epsilon} \qquad \frac{}{C \vdash \mathbf{drop} : t \to \epsilon} \qquad \frac{}{C \vdash \mathbf{select} : t\ t\ \mathsf{i32} \to t}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\,(t_2^m) \vdash e^* : tf}{C \vdash \mathbf{block}\ tf\ e^*\ \mathbf{end} : tf} \qquad \frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\,(t_1^n) \vdash e^* : tf}{C \vdash \mathbf{loop}\ tf\ e^*\ \mathbf{end} : tf}$$

$$\frac{tf = t_1^n \to t_2^m \qquad C, \mathsf{label}\,(t_2^m) \vdash e_1^* : tf \qquad C, \mathsf{label}\,(t_2^m) \vdash e_2^* : tf}{C \vdash \mathbf{if}\ tf\ e_1^*\ \mathbf{else}\ e_2^*\ \mathbf{end} : t_1^n\ \mathsf{i32} \to t_2^m}$$

$$\frac{C_{\mathsf{label}}(i) = t^*}{C \vdash \mathbf{br}\ i : t_1^*\ t^* \to t_2^*} \qquad \frac{C_{\mathsf{label}}(i) = t^*}{C \vdash \mathbf{br\_if}\ i : t^*\ \mathsf{i32} \to t^*} \qquad \frac{(C_{\mathsf{label}}(i) = t^*)^+}{C \vdash \mathbf{br\_table}\ i^+ : t_1^*\ t^*\ \mathsf{i32} \to t_2^*}$$

$$\frac{C_{\mathsf{return}} = t^*}{C \vdash \mathbf{return} : t_1^*\ t^* \to t_2^*} \qquad \frac{C_{\mathsf{func}}(i) = tf}{C \vdash \mathbf{call}\ i : tf} \qquad \frac{tf = t_1^* \to t_2^* \qquad C_{\mathsf{table}} = n}{C \vdash \mathbf{call\_indirect}\ tf : t_1^*\ \mathsf{i32} \to t_2^*}$$

$$\frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{get\_local}\ i : \epsilon \to t} \qquad \frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{set\_local}\ i : t \to \epsilon} \qquad \frac{C_{\mathsf{local}}(i) = t}{C \vdash \mathbf{tee\_local}\ i : t \to t} \qquad \frac{C_{\mathsf{global}}(i) = \mathsf{mut}^?\ t}{C \vdash \mathbf{get\_global}\ i : \epsilon \to t} \qquad \frac{C_{\mathsf{global}}(i) = \mathsf{mut}\ t}{C \vdash \mathbf{set\_global}\ i : t \to \epsilon}$$

$$\frac{C_{\mathsf{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad (tp\_sz)^? = \epsilon \vee t = \mathsf{i}m}{C \vdash t.\mathbf{load}\ (tp\_sz)^?\ a\ o : \mathsf{i32} \to t} \qquad \frac{C_{\mathsf{memory}} = n \qquad 2^a \leq (|tp| <)^? |t| \qquad tp^? = \epsilon \vee t = \mathsf{i}m}{C \vdash t.\mathbf{store}\ tp^?\ a\ o : \mathsf{i32}\ t \to \epsilon}$$

$$\frac{C_{\mathsf{memory}} = n}{C \vdash \mathbf{current\_memory} : \epsilon \to \mathsf{i32}} \qquad \frac{C_{\mathsf{memory}} = n}{C \vdash \mathbf{grow\_memory} : \mathsf{i32} \to \mathsf{i32}}$$

$$\frac{}{C \vdash \epsilon : \epsilon \to \epsilon} \qquad \frac{C \vdash e_1^* : t_1^* \to t_2^* \qquad C \vdash e_2 : t_2^* \to t_3^*}{C \vdash e_1^*\ e_2 : t_1^* \to t_3^*} \qquad \frac{C \vdash e^* : t_1^* \to t_2^*}{C \vdash e^* : t^*\ t_1^* \to t^*\ t_2^*}$$

**Typing Modules**

$$\frac{tf = t_1^* \to t_2^* \qquad C, \mathsf{local}\ t_1^*\ t^*, \mathsf{label}\,(t_2^*), \mathsf{return}\,(t_2^*) \vdash e^* : \epsilon \to t_2^*}{C \vdash ex^*\ \mathbf{func}\ tf\ \mathbf{local}\ t^*\ e^* : ex^*\ tf} \qquad \frac{tg = \mathsf{mut}^?\ t \qquad C \vdash e^* : \epsilon \to t \qquad ex^* = \epsilon \vee tg = t}{C \vdash ex^*\ \mathbf{global}\ tg\ e^* : ex^*\ tg}$$

$$\frac{(C_{\mathsf{func}}(i) = tf)^n}{C \vdash ex^*\ \mathbf{table}\ n\ i^n : ex^*\ n} \qquad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n : ex^*\ n}$$

$$\frac{}{C \vdash ex^*\ \mathbf{func}\ tf\ im : ex^*\ tf} \qquad \frac{tg = t}{C \vdash ex^*\ \mathbf{global}\ tg\ im : ex^*\ tg} \qquad \frac{}{C \vdash ex^*\ \mathbf{table}\ n\ im : ex^*\ n} \qquad \frac{}{C \vdash ex^*\ \mathbf{memory}\ n\ im : ex^*\ n}$$

$$\frac{\begin{array}{c} (C \vdash f : ex_{\mathrm{f}}^*\ tf)^* \qquad (C_i \vdash glob_i : ex_{\mathrm{g}}^*\ tg_i)_i^* \qquad (C \vdash tab : ex_{\mathrm{t}}^*\ n)^? \qquad (C \vdash mem : ex_{\mathrm{m}}^*\ n)^? \\ (C_i = \{\mathsf{global}\ tg^{i-1}\})_i^* \qquad C = \{\mathsf{func}\ tf^*, \mathsf{global}\ tg^*, \mathsf{table}\ n^?, \mathsf{memory}\ n^?\} \qquad ex_{\mathrm{f}}^{*\,*}\ ex_{\mathrm{g}}^{*\,*}\ ex_{\mathrm{t}}^{*\,?}\ ex_{\mathrm{m}}^{*\,?}\ \text{distinct} \end{array}}{\vdash \mathbf{module}\ f^*\ glob^*\ tab^?\ mem^?}$$

**Figure 3.** Typing rules

$$\text{(store context)} \quad S \quad ::= \quad \{\mathsf{inst}\ C^*,\ \mathsf{tab}\ n^*,\ \mathsf{mem}\ n^*\}$$

$$\frac{S = \{\mathsf{inst}\ C^*,\ \mathsf{tab}\ n^*,\ \mathsf{mem}\ m^*\} \qquad (S \vdash inst : C)^* \qquad ((S \vdash cl : tf)^*)^* \qquad (n \leq |cl^*|)^* \qquad (m \leq |b^*|)^*}{\vdash \{\mathsf{inst}\ inst^*,\ \mathsf{tab}\ (cl^*)^*,\ \mathsf{mem}\ (b^*)^*\} : S}$$

$$\frac{S_{\mathsf{inst}}(i) \vdash f : tf}{S \vdash \{\mathsf{inst}\ i,\ \mathsf{code}\ f\} : tf} \qquad \frac{(S \vdash cl : tf)^* \qquad (\vdash v : t)^* \qquad (S_{\mathsf{tab}}(i) = n)^? \qquad (S_{\mathsf{mem}}(j) = m)^?}{S \vdash \{\mathsf{func}\ cl^*,\ \mathsf{glob}\ v^*,\ \mathsf{tab}\ i^?,\ \mathsf{mem}\ j^?\} : \{\mathsf{func}\ tf^*,\ \mathsf{global}\ (\mathsf{mut}^?\ t)^*,\ \mathsf{table}\ n^?,\ \mathsf{memory}\ m^?\}}$$

$$\frac{\vdash s : S \qquad S; \epsilon \vdash_i v^*; e^* : t^*}{\vdash_i s; v^*; e^* : t^*} \qquad \frac{(\vdash v : t_v)^* \qquad S; S_{\mathsf{inst}}(i), \mathsf{local}\ t_v^*, \mathsf{return}\ (t^*)^? \vdash e^* : \epsilon \to t^*}{S; (t^*)^? \vdash_i v^*; e^* : t^*} \qquad \frac{}{\vdash t.\mathbf{const}\ c : t}$$

$$\frac{}{C \vdash \mathbf{trap} : tf} \qquad \frac{C \vdash e_0^* : t_1^n \to t_2^* \qquad C, \mathsf{label}\ (t_1^n) \vdash e^* : \epsilon \to t_2^*}{C \vdash \mathbf{label}_n\{e_0^*\}\ e^*\ \mathbf{end} : \epsilon \to t_2^*} \qquad \frac{S \vdash cl : tf}{S; C \vdash \mathbf{call}\ cl : tf} \qquad \frac{S; (t^n) \vdash_i v^*; e^* : t^n}{S; C \vdash \mathbf{local}_n\{i; v^*\}\ e^*\ \mathbf{end} : \epsilon \to t^n}$$

**Figure 4.** Store and configuration typing and rules for administrative instructions

# Semantics

- Statically as a type system (“embarassingly simple”)
- Dynamically as a nondeterministic, relational small-step reduction, and types extending the static setting.
- Combining these two, soundness provides:
  - Type Safety (locals, globals, instruction/function args)
  - Memory Safety (locals, globals, tables, memory)
  - No undefined behaviour (evaluation rules cover all cases and are mutually consistent)
  - Encapsulation (scope of locals, module components according to imports/exports)

## Soundness: Key Theorems

**Preservation** If $\vdash (S, T) : t$ and $(S, T) \hookrightarrow (S', T')$ then $\vdash (S', T') : t$ and $\vdash S \preceq S'$.

**Progress** If $\vdash (S, T) : t$ then either $(S, T)$ is terminal or there is $(S', T')$ s.t. $(S, T) \hookrightarrow (S', T')$

**Soundness** If $\vdash (S, T) : t$ then $(S, T)$ either diverges or takes a finite number of $\hookrightarrow$-steps to reach a terminal configuration $(S', T')$ s.t. $\vdash (S', T') : t$

Every program either runs forever, traps, or terminates. Proofs mechanized in Isabelle/HOL in [2] based on the definitions on paper in [1].

# Maturity and Adoption I

- Specification officially (still) a draft, however widely implemented and used.
- Released a minimum viable product early on.
- Formal specification, reference interpreter (OCaml) maintained by spec authors.
- Many embedders in/compilers from various languages available (just search GitHub).
- Early Adopters: Gamers! Run all the flash minigames with WebAssembly.
- But now, it is possible to compile game engines from C/C++ to the web!
- “Native performance on the web.”
- Leveraging legacy codebases, i.e. PSPDFKit’s PDF handling library.
- Standalone VM using LLVM IR and JIT

# Future

- Multiple memories, 64bit addressable memory, multiple return values
- Garbage collection
- Concurrency through threads
- Exception handling
- Tail Calls
- SIMD

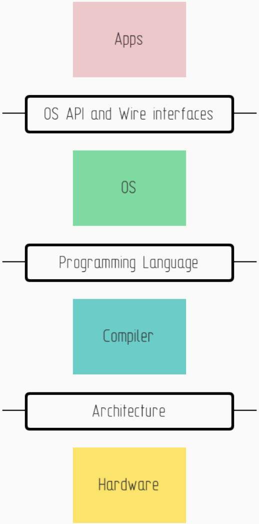
# Demos

- AngryBots (Unity game engine)
- amazeballz
- PSPDFKit for Web
- Real world benchmark based on PSPDFKit for Web

CAKEML

# REMS

**Rigorous Engineering of Mainstream Systems**




**Task 4a: System APIs and Protocols**

TLS: nqsbTLS
TCP: Huginn-TCP
POSIX filesystem test oracle: SibylFS
POSIX filesystem logic

**Task 3: Systems Programming Languages**

Sequential C (de facto/ISO/CHERI): Cerberus
Concurrent C: C/C++11 and new models
C runtime typechecking: libcrunch
ELF linking: linksem
Verified ML implementation: CakeML
JavaScript

**Task 2: Architectural Multiprocessor Semantics**

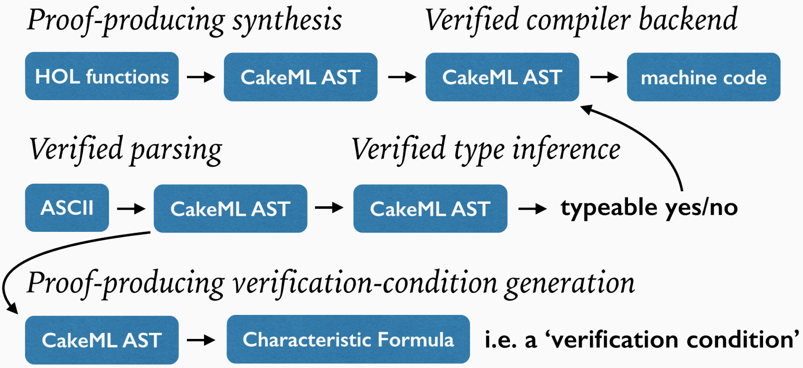
Sail ISA models: ARM, Power, MIPS, CHERI
L3 ISA models: ARM, MIPS, x86, CHERI, RISC-V
ARM and Power Concurrency
CHERI ISA

*"Task 1: Semantic tools" and*
*"Task 4b: Concurrency Reasoning" not shown.*

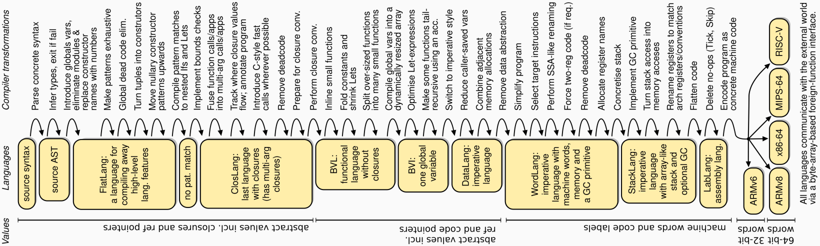
# A verified Implementation of ML

- ...as a compiler in Higher Order Logic.
- First verified compiler of a functional programming language (POPL'14).
- Developed by team of 16+25 with many publications, please see website!
- CakeML is a substantial subset of Standard ML.
- Two frontends:
    1. Translate from HOL to CakeML abstract syntax
    2. Parse CakeML concrete syntax
- Optimizing backend which targets x86, ARM, RISC-V, MIPS. Working on targeting WebAssembly.
- Can bootstrap inside HOL, i.e. compile itself via frontend 1.
- Allows calls to a foreign function interface

# Ecosystem


Proof-producing synthesis
Verified compiler backend
HOL functions
CakeML AST
CakeML AST
machine code
Verified parsing
Verified type inference
ASCII
CakeML AST
CakeML AST
typeable yes/no
Proof-producing verification-condition generation
CakeML AST
Characteristic Formula
i.e. a ‘verification condition’

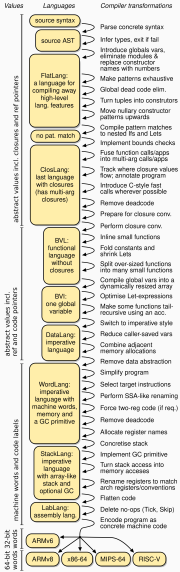
# Compiler Architecture



# Translating CakeML to WebAssembly

# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

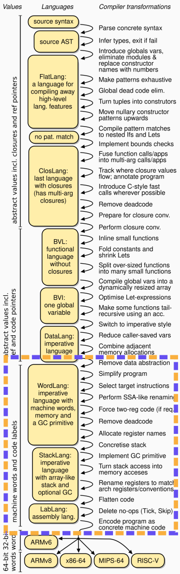



All languages communicate with the external world via a byte-array-based foreign-function interface.

# Which intermediate language to compile from?

Looking for a language that matches WebAssembly’s features:

1. Operates on words

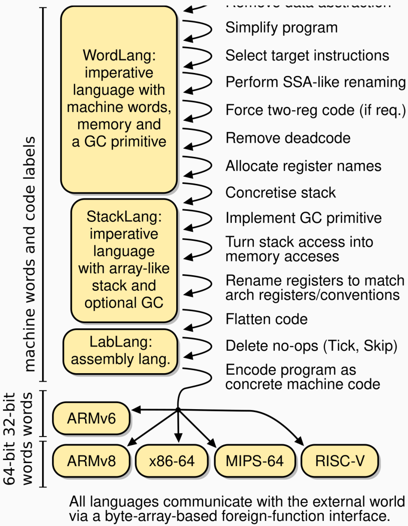



All languages communicate with the external world via a byte-array-based foreign-function interface.

# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

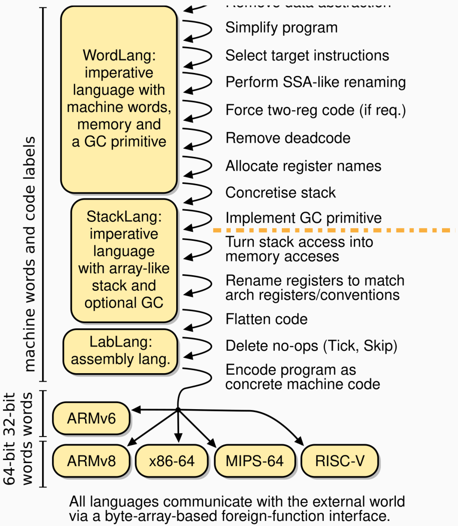
1. Operates on words



# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

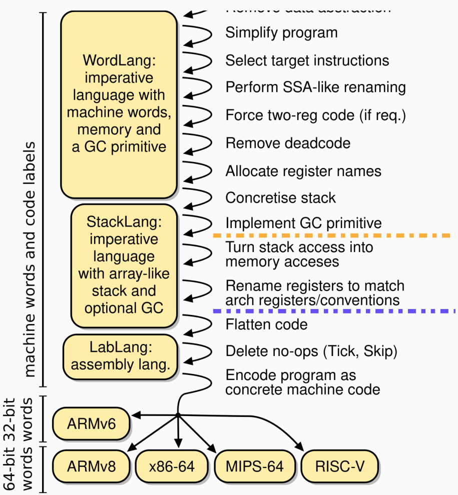
1. Operates on words
2. Manages memory



# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

1. Operates on words
2. Manages memory
3. Local control flow

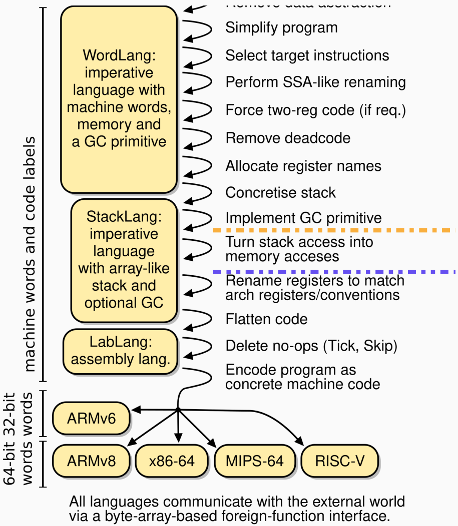



All languages communicate with the external world via a byte-array-based foreign-function interface.

# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

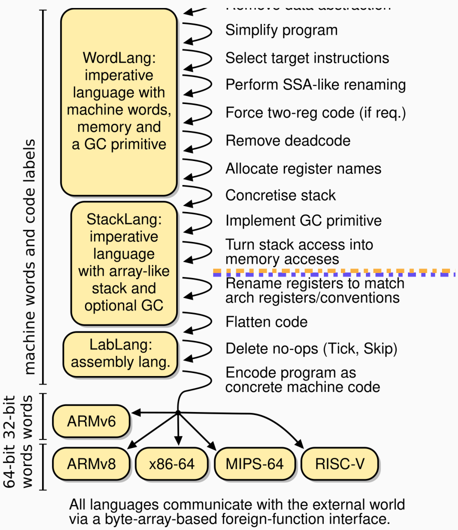
1. Operates on words
2. 🟧 Manages memory
3. 🟪 Local control flow
4. 🟪 No register usage conventions



# Which intermediate language to compile from?

Looking for a language that matches WebAssembly's features:

1. Operates on words
2. ■ Manages memory
3. ■ Local control flow
4. ■ No register usage conventions
5. ■ Implicit stack?



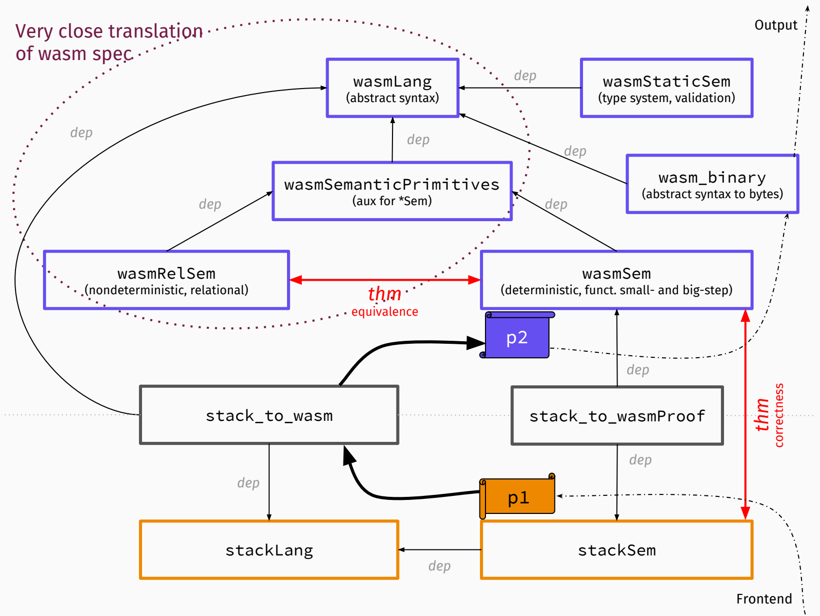
# Challenges ≈ mismatch(StackLang, WebAssembly)

What I need to “compile away”:

- No exception handling.
- No tail recursive calls in WebAssembly!
    1. Cannot use `call`/`return` without risk of stack overflows.
    2. Hence, emulate jumps in WebAsssembly!
    3. Jumps emulated by `br_table` which with local control flow.
    4. Cannot use `if`, `loop`, `block`, `br`.

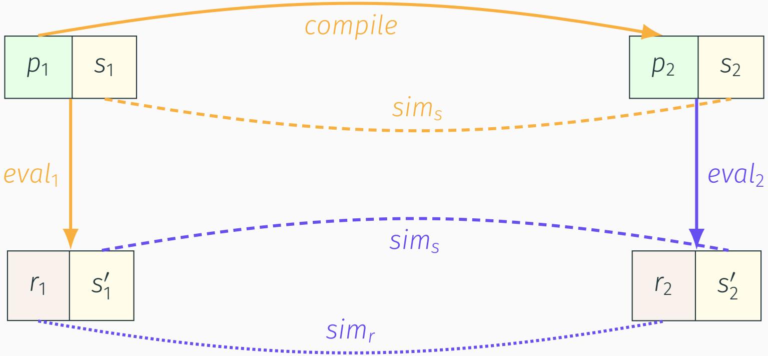
Also: Provide a functional big-step semantics.

# Implementation



# Verification

# Compiler Correctness Theorem




$$(\forall \ \ldots) \ \ eval_1 \, p_1 \, s_1 = (r_1, s'_1) \land sim_s \, s_1 \, s_2 \land compile \, p_1 = p_2 \land r_1 \neq Error$$

$$\longrightarrow$$

$$(\exists \, r_2 \, s'_2) \ \ eval_2 \, p_2 \, s_2 = (r_2, s'_2) \land sim_s \, s'_1 \, s'_2 \land sim_r \, r_1 \, r_2$$

# Progress and Outlook

# Progress

- Mechanized static and dynamic semantics
- Mechanized most of the initialization semantics
- Specified an alternative dynamic semantics in functional big-step style
- Prototyped translation
- Prepared the CakeML compiler architecture for integrating the new target
- Proved some lemmas, mostly about the relation between the two semantics
- Thought about and formulated some …conjectures …
- Prototyped a runtime to execute generated code in browser

# TODO

- Prove that all compiler output is valid WebAssembly
- Prove compiler correctness

# Recap

Introduction

- WebAssembly
- CakeML

Translating CakeML to WebAssembly

Verification

Progress and Outlook

Questions, please!

# References I


A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. F. Bastien.
**Bringing the web up to speed with webassembly.**
In A. Cohen and M. T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 185–200. ACM, 2017.

C. Watt.
**Mechanising and verifying the webassembly specification.**
In J. Andronick and A. P. Felty, editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 53–65. ACM, 2018.