VeriGHC - Separation Logic for GHC's Cmm Language

Peter Trommler

Faculty of Computer Science
Technische Hochschule Nürnberg

Oberseminar Theoretische Informatik, Nov 20, 2018

Outline

GHC compiler pipeline

Where Cmm fits in

- Haskell
- Core (System F with Coercions)
- STG (Shared Term Graph-reduction or Spineless Tagless G-machine)
- Cmm
- LLVM IR or assembly or C

Motivation

- Reasoning about concurrency
- Integrate with correctness proofs of run-time system
- Prove correctness of Cmm optimizations
- Extract a certified GHC back-end for PowerPC 64-bit

CompCert and the Verified Software Toolchain

- CompCert by Xavier Leroy et al.
  - C style, byte addressable memory model
  - Formalisation of integer arithmetic modulo 2^n
  - Formalisation of instruction set for Intel, Arm, and PowerPC
- Verified Software Toolchain by Andrew Appel et al.
  - Mechanized Semantics Library: Separation Logic
  - Proof automation for separation logic
  - Omega-like tactic for modulo arithmetic

Cmm: GHC's C– – dialect

- C– –, language for compiler backends with informal specification
- Cminor, intermediate language in CompCert "inspired by C"
- Cmm, intermediate language in GHC; parts of RTS written in Cmm
- Cmm AST, let's call it Cmm Core
  - High-level portable assembly
  - Expressions
  - Statements (Nodes)
  - Procedures and Program

Syntax of Expressions

e	$::=$	v	constants
	$\mid$	$[e]_\tau$	read memory location
	$\mid$	x	contents of register
	$\mid$	$\bigcirc\, \overline{e}$	machine operation
	$\mid$	$[sp_{area} + \delta]$	(read from stack)
	$\mid$	$\rho + \delta$	(register + offset)

- items in parentheses will not be covered in this talk

Values

$$
\begin{array}{lcll}
v & ::= & \mathit{HSundef} & \text{undefined value} \\
 & | & \mathit{HSint}(i) & \text{integer (64 bit)} \\
 & | & \mathit{HSsingle}(f) & \text{IEEE single precision float} \\
 & | & \mathit{HSdouble}(d) & \text{IEEE double precision float} \\
 & | & \mathit{HSptr}(b, i) & \text{Pointer to block } b \text{ of length } i
\end{array}
$$

Ψ	program with global environment
sp	stack pointer, current activation record
ρ	local environment, mapping identifiers to values
ϕ	memory footprint, map of permissions, for separation logic
m	heap

Expressions: Big-Step Semantics

$$\Psi;\ (sp; \rho; \phi; m) \vdash v \Downarrow \text{haskell\_value(v)}$$

$$\frac{x \in \text{dom}\,\rho}{\Psi;\ (sp; \rho; \phi; m) \vdash x \Downarrow \rho(x)}$$

$$\frac{\Psi;\ (sp; \rho; \phi; m) \vdash \overline{e} \Downarrow \overline{v} \qquad \Psi;\ sp \vdash \bigcirc\,\overline{v} \Downarrow_{eval\_operation} v}{\Psi;\ (sp; \rho; \phi; m) \vdash \bigcirc\,\overline{e} \Downarrow v}$$

$$\frac{\Psi;\ (sp; \rho; \phi; m) \vdash e_1 \Downarrow v_1 \qquad \phi \vdash \text{load}_\tau v_1 \qquad m \vdash v_1 \mapsto v}{\Psi;\ (sp; \rho; \phi; m) \vdash [e_1]_\tau \Downarrow v}$$

Syntax of Statements (CmmNode)

s	$::=$	$\ell:$	label
	$\mid$	$\{-\text{text}-\}$	comment
	$\mid$	$\rho := e$	assign register
	$\mid$	$[e]_\tau := e$	store in memory
	$\mid$	$\textbf{goto}\ \ell$	branch
	$\mid$	$\textbf{if}\ e\ \textbf{then goto}\ \ell\ \textbf{else goto}\ \ell$	conditional branch
	$\mid$	$\textbf{switch}\ e\ \overline{(i, \ell)}$	jump table
g	$::=$	$\ell : \overline{s}$	labelled graph of stmts.

C while statement in Cmm

while (e) {
s_1
continue;
s_2
break;
s_3
}

loop:
if e **then** **goto** body **else** **goto** end
body:
s'_1
goto loop
s'_2
goto end
s'_3
goto loop
end:

Continuations

- Control stack

$$\kappa ::= \text{Kstop} \mid s \cdot \kappa \mid \text{Kcall } \overline{x} \; f \; sp \; \rho \; \kappa$$

- Continuation

$$k ::= (\sigma, \kappa)$$

- State

$$\sigma := (sp; \rho; \phi; m)$$

- Notation:

$$\Psi; \sigma \vdash e \Downarrow v$$

$$\Psi; (sp_\sigma; \rho_\sigma; \phi_\sigma; m_\sigma) \vdash e \Downarrow v$$

Small-Step Semantics

$$\Psi \vdash (\sigma, (s_1 ; s_2) \cdot \kappa) \longmapsto (\sigma, s_1 \cdot s_2 \cdot \kappa)$$

$$\frac{\Psi; \sigma \vdash e \Downarrow v \qquad \rho' = \rho_\sigma[x := v]}{\Psi \vdash (\sigma, (x := e) \cdot \kappa) \longmapsto (\sigma[:= \rho'], \kappa)}$$

$$\frac{\Psi; \sigma \vdash e_1 \Downarrow v_1 \qquad \Psi; \sigma \vdash e_2 \Downarrow v_2 \qquad \tau = \text{typeof } e_2 \\ \phi_\sigma \vdash \text{store}_\tau v_1 \qquad m' = m_\sigma[v_1 :=_\tau v_2]}{\Psi \vdash (\sigma, ([e_1]_\tau := e_2) \cdot \kappa) \longmapsto (\sigma[:= m'], \kappa)}$$

$$\Psi \vdash (\sigma, \ell : \cdot \kappa) \longmapsto (\sigma, \kappa)$$

$$\Psi \vdash (\sigma, \{-\text{text}-\} \cdot \kappa) \longmapsto (\sigma, \kappa)$$

Small-Step Semantics: Control

$$\frac{j \geq 1 \qquad s_j \text{ jumpish} \qquad s_1, \ldots, s_{j-1} \text{ not jumpish}}{\Psi \vdash (\sigma, \textbf{goto } \ell \cdot \kappa) \longmapsto (\sigma, \ell :\cdot s_1 \ldots s_j \cdot \kappa)}$$

$$\frac{\Psi; \sigma \vdash e \Downarrow v \qquad \text{is\_true } v}{\Psi \vdash (\sigma, \textbf{if } e \textbf{ then goto } \ell_1 \textbf{ else goto } \ell_2 \cdot \kappa) \longmapsto (\sigma, \textbf{goto } \ell_1 \cdot \kappa)}$$

$$\frac{\Psi; \sigma \vdash e \Downarrow v \qquad \text{is\_false } v}{\Psi \vdash (\sigma, \textbf{if } e \textbf{ then goto } \ell_1 \textbf{ else goto } \ell_2 \cdot \kappa) \longmapsto (\sigma, \textbf{goto } \ell_2 \cdot \kappa)}$$

$$\frac{\Psi; \sigma \vdash e \Downarrow v \qquad v = i}{\Psi \vdash (\sigma, \textbf{switch } e\ \overline{(i, \ell_i)} \cdot \kappa) \longmapsto (\sigma, \textbf{goto } \ell_i \cdot \kappa)}$$

Separation Logic: Definitions

- pure(e): e does not read from memory
- $e \Downarrow v$: emp $\wedge$ pure(e) $\wedge$ $\lambda\Psi\sigma.(\Psi;\sigma \vdash e \Downarrow v)$
- defined(e): e is not undefined (*HSundef*)

SeparationLogic: Hoare Sextuple

- Γ: properties of the global environment
- R: function's postcondition as predicate on list of returned values
- B: precondition of each labelled statement block (ℓ)
- $\{P\}c$: P guards c, in all states σ it is safe to execute c
- $\{P\}c\{Q\}$: $\forall k.\{Q\}k \rightarrow \{P\}(c; k)$

Separation Logic: Label, Comment, Consequence, and Sequence

$$\Gamma; R; B \vdash \{P\}\ell : \{P\}$$

$$\Gamma; R; B \vdash \{P\}\{-\text{text}-\}\{P\}$$

$$\frac{P \Rightarrow P' \qquad \Gamma; R; B \vdash \{P'\}s\{Q'\} \qquad Q' \Rightarrow Q}{\Gamma; R; B \vdash \{P\}s\{Q\}}$$

$$\frac{\Gamma; R; B \vdash \{P\}s_1\{P'\} \qquad \Gamma; R; B \vdash \{P'\}s_2\{Q\}}{\Gamma; R; B \vdash \{P\}s_1 ; s_2\{Q\}}$$

Separation Logic: Assignment and Store

$$\frac{\rho' = \rho_\sigma[x := v] \qquad P = (\exists v. e \Downarrow v \land \lambda\sigma. Q\ \sigma[:= \rho'])}{\Gamma; R; B \vdash \{P\} x := e \{Q\}}$$

$$\frac{\text{pure}(e) \qquad \text{pure}(e_2) \qquad P = (e \mapsto_\tau e_2 * \text{defined}(e_1))}{\Gamma; R; B \vdash \{P\} [e]_\tau := e_1 \{e \mapsto_\tau e_1\}}$$

Separation Logic: Control

$$\Gamma; R; B \vdash \{B(\ell)\}\textbf{goto } \ell\{\bot\}$$

$$\frac{\text{pure}(e) \qquad \Gamma; R; B \vdash \{P \wedge e\}\textbf{goto } \ell_1\{\bot\} \qquad \Gamma; R; B \vdash \{P \wedge \neg e\}\textbf{goto } \ell_2\{\bot\}}{\Gamma; R; B \vdash \{P\}\textbf{if } e \textbf{ then goto } \ell_1 \textbf{ else goto } \ell_2\{\bot\}}$$

$$\frac{\text{pure}(e) \qquad \Gamma; R; B \vdash \{P \wedge e = i\}\textbf{goto } \ell_i\{\bot\}}{\Gamma; R; B \vdash \{P\}\textbf{switch } e\ \overline{(i, \ell_i)}\{\bot\}}$$

Separation Logic: Frame Rule

$$\frac{\Gamma; R; B \vdash \{P\}s\{Q\} \qquad \text{modified vars}(s) \cap \text{free vars}(A) = \emptyset}{\Gamma; (\lambda\overline{v}.A * R(\overline{v})); (\lambda\ell.A * B(\ell)) \vdash \{A * P\}s\{A * Q\}}$$

Conclusion and Future Work

- Small-Step semantics and separation logic for Cmm Core statements
- Formalize in Coq
- Proof automation (from MSL)
- Write PowerPC backend and extract code
- Cmm to Coq compiler (like clightgen)
- Verify concurrency features of RTS using formalization of POWER memory consistency model by Sewell et al.