A more precise, more correct stack and register model for CompCert

Gergö Barany

Inria Paris, France
Optimizing C compiler, written and verified in Coq

Simulation proofs of semantic preservation

\[ S_1 \sim C_1 \]

\[ t \]

\[ \vdots \]

\[ S_2 \sim C_2 \]
Values and types in CompCert

**Values**

*Inductive* \( \text{val} := \) Vundef

*Inductive* \( \text{typ} := \) Tany32 (* int, single *)

| Tany64 (* long, float *)

‘Less defined’ relation

\[ v1 \leq_{\text{def}} v2 \iff v1 = \text{Vundef} \lor v1 = v2 \]
Registers and the stack in CompCert

**Locations: registers and stack slots**

*Inductive* \( \text{mreg} := \text{R0} \mid \text{R1} \mid \text{R2} \mid ... \)

*Inductive* \( \text{loc} := \text{R} (r: \text{mreg}) \mid \text{S} (sl: \text{slot}) (\text{pos}: \text{Z}) (\text{ty}: \text{typ}) \).

**State of local variables: values of such locations**

*Definition* \( \text{locmap} := \text{loc} \rightarrow \text{val} \).

*Definition* \( \text{get} (l: \text{loc}) (m: \text{locmap}) := m l \).

*Definition* \( \text{set} (l: \text{loc}) (v: \text{val}) (m: \text{locmap}): \text{locmap} := ... \)

**Example property: the locmap remembers values**

*Lemma* \( \text{gss\_reg} \):

\[
\forall (r: \text{mreg}) (v: \text{val}) (m: \text{locmap}), \text{get} (\text{R} r) (\text{set} (\text{R} r) v m) = v.
\]
Motivation: Subregister aliasing

‘Big’ registers as pairs of ‘small’ ones

ARM floating-point registers:

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>⋮</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
</tbody>
</table>

Kalray MPPA general-purpose registers:

<table>
<thead>
<tr>
<th>R0R1</th>
<th>R2R3</th>
<th>R4R5</th>
<th>R6R7</th>
<th>⋮</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
<td>R4</td>
</tr>
</tbody>
</table>

Writes to a superregister invalidate the subregisters and vice versa

Example (ARM):

```
vmov.f64 d0, #1.0e+0  @ double x = 1.0;
vmov.f32 s0, #2.0e+0  @ float y = 2.0f;
... @ d0 is now garbled (undefined)
```
First try: Invalidate aliasing registers

**Definition** `set_reg` \((r: mreg)(v: val)(m: locmap): locmap :=
let \(m' :=\) undefined aliasing registers \(r m\) in
fun \(loc \Rightarrow\)
  match \(loc\) with
  | \(R r' \Rightarrow\) if \(r' = r\) then \(v\) else \(m' r'\)
  | \(S _ _ _ \Rightarrow m' loc\)
end.

+ Models everything we need to know about registers

**Problem: Spilling**

– Local variables must sometimes be spilled to the stack
– Restoring aliased registers impossible in this setting:
  restoring subregisters invalidates superregister and vice versa
Second try: Pairs of word-sized values

**Inductive** word := Wundef | Wint (i: int) | Wsingle (s: float32).

**Inductive** val := ... (* as before *) | Vpair (lo hi: word).

- Subregister accesses read/write halves of a `Vpair`
- A pair can be spilled as a unit and will be restored correctly

**Problem: ‘Less defined’ relation**

We now have, for example:

\[ Vundef \leq_{\text{def}} Vpair (Wint i)(Wundef) \leq_{\text{def}} Vpair (Wint i)(Wint j) \]

- Breaks invariant: \( v1 \leq_{\text{def}} v2 \iff v1 = Vundef \lor v1 = v2 \)
- Breaks all the proofs
Third try: Registers and stack slots as blocks of bytes

Model register file as block of bytes ‘addressed’ by registers

<table>
<thead>
<tr>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>S6</td>
<td>S7</td>
</tr>
</tbody>
</table>

(and similarly for stack slots)
Encoding and decoding values

**Byte-sized values (also used for heap memory)**

**Inductive** `quantity := Q32 | Q64`.

**Inductive** `memval :=
- Undef: memval
- Fragment (v: val) (q: quantity) (n: nat)
- Byte (b: byte) (* heap memory only *).

**Encode/decode using functions** `inj_value, proj_value`

`inj_value Q32 (Vint 42) =
  Fragment (Vint 42) Q32 3 :: Fragment (Vint 42) Q32 2 ::
  Fragment (Vint 42) Q32 1 :: Fragment (Vint 42) Q32 0 :: nil`

`proj_value Q32
  (Fragment (Vint 42) Q32 1 :: Fragment (Vsingle 0.0) Q32 2 :: nil) = Vundef`
Accessing registers as blocks of bytes

Access model

Definition get_bytes (r: mreg) (rf: regfile): list memval := ...
Definition set_bytes (r: mreg) (bs: list memval) (rf: regfile) ...

Definition get (r: mreg) (rf: regfile): val :=
   proj_value (quantity_of_mreg r) (get_bytes r rf).
Definition set (r: mreg) (v: val) (rf: regfile): regfile :=
   set_bytes r (inj_value (quantity_of_mreg r) v) rf.

- ‘normal’ accesses encode/decode values to bytes on the fly
- copy/spill/reload operations copy raw bytes
+ writes invalidate aliasing registers
+ correctness of spilling is provable
Agreement of program states

Simulation proofs use $\sim$ (agreement) relation

Definition agree state_s state_c :=
    $\forall (l: \text{loc}), (\text{get } l \text{ state}_s) \leq_{\text{def}} (\text{get } l \text{ state}_c)$.

(agreement on values)

Need a stronger notion of byte-wise agreement

‘less defined’ relation on bytes:

$$b_1 \leq_{\text{def,byte}} b_2 \iff b_1 = \text{Undef} \lor b_1 = b_2$$

Definition agree_bytes state_s state_c :=
    $\forall (l: \text{loc}), \text{list_forall2} \leq_{\text{def,byte}} (\text{get_bytes } l \text{ state}_s)$
    $\quad (\text{get_bytes } l \text{ state}_c)$.

agreement on bytes; implies agreement on values
Problem: Type safety of register accesses

**The gss_reg lemma revisited**

**Lemma gss_reg**: 
\[ \forall (r: \text{mreg}) (v: \text{val}) (m: \text{locmap}), \text{get} (R r) (\text{set} (R r) v m) = v. \]

No longer provable... but also incorrect:

**Example gss_reg_bad**: 
\[ \forall m, \text{get} (R S0) (\text{set} (R S0) (\text{Vfloat DBL_MAX}) m) = \text{Vfloat DBL_MAX}. \]

(storing 64 bits in a 32-bit register and getting them back)

**Correct formulation**

**Lemma gss_reg**: 
\[ \forall (r: \text{mreg}) (v: \text{val}) (m: \text{locmap}), \text{Val.\_has\_type} v (\text{mreg\_type} r) \rightarrow \text{get} (R r) (\text{set} (R r) v m) = v. \]

**tedious**: must prove well-typedness at every use of the lemma
Insufficient checks in \texttt{inj\_value} and \texttt{proj\_value}

\begin{verbatim}
inj_value Q32 \ (Vlong 18446744073709551615) = Fragment \ (Vlong 18446744073709551615) Q32 3
  :: Fragment \ (Vlong 18446744073709551615) Q32 2
    :: Fragment \ (Vlong 18446744073709551615) Q32 1
      :: Fragment \ (Vlong 18446744073709551615) Q32 0 :: nil

proj_value Q32 (*the above*) = Vlong 18446744073709551615
\end{verbatim}

(easy to fix)
Reworked CompCert’s register and stack model

- more precise: prove properties of subregister aliasing, spilling
- more correct: badly typed register accesses no longer allowed
- even nice, simple, ‘obviously correct’ models hard to get right

Thank you for your attention

This research was partially supported by ITEA 3 project no. 14014, ASSUME.