

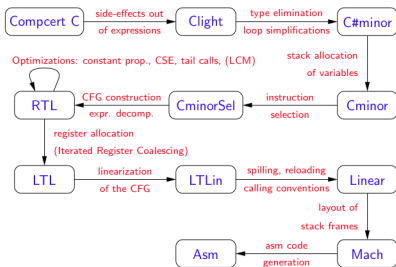
# **A more precise, more correct stack and register model for CompCert**

---

Gergö Barany

Inria Paris, France

## Optimizing C compiler, written and verified in Coq



## Simulation proofs of semantic preservation

$$\begin{array}{ccc}
 S_1 & \overset{\sim}{\text{---}} & C_1 \\
 t \downarrow & & \vdots t \\
 S_2 & \cdots \overset{\sim}{\text{---}} \cdots & C_2
 \end{array}$$

# Values and types in CompCert

## Values

```
Inductive val := Vundef
  | Vint (i: int)
  | Vlong (l: int64)
  | Vsingle (s: float32)
  | Vfloat (f: float64)
  | Vptr (...).
```

## Types

```
Inductive typ := Tany32 (* int, single *)
  | Tany64 (* long, float *)
```

## 'Less defined' relation

$$v1 \leq_{\text{def}} v2 \iff v1 = \text{Vundef} \vee v1 = v2$$

# Registers and the stack in CompCert

## Locations: registers and stack slots

**Inductive** `mreg` := R0 | R1 | R2 | ...

**Inductive** `loc` := R (r: mreg) | S (sl: slot) (pos: Z) (ty: typ).

## State of local variables: values of such locations

**Definition** `locmap` := `loc` → `val`.

**Definition** `get` (l: loc) (m: locmap) := m l.

**Definition** `set` (l: loc) (v: val) (m: locmap): locmap := ...

## Example property: the `locmap` remembers values

**Lemma** `gss_reg`:

$\forall (r: \text{mreg}) (v: \text{val}) (m: \text{locmap}), \text{get } (R\ r) (\text{set } (R\ r)\ v\ m) = v.$

## Motivation: Subregister aliasing

### 'Big' registers as pairs of 'small' ones

ARM floating-point registers:

D0		D1		D2		D3		...	
S0	S1	S2	S3	S4	S5	S6	S7	...	

Kalray MPPA general-purpose registers:

R0R1		R2R3		R4R5		R6R7		...	
R0	R1	R2	R3	R4	R5	R6	R7	...	

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' := undef_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

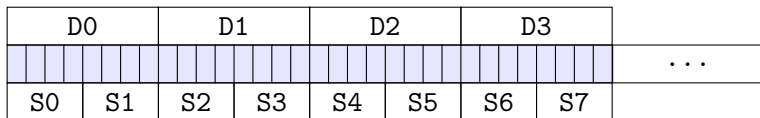
We now have, for example:

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

- Breaks invariant:  $v1 \leq_{\text{def}} v2 \iff v1 = Vundef \vee v1 = v2$
- Breaks all the proofs

## Third try: Registers and stack slots as blocks of bytes

Model register file as block of bytes 'addressed' by registers



(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:

$$b1 \leq_{\text{def,byte}} b2 \iff b1 = \text{Undef} \vee b1 = b2$$

**Definition** agree\_bytes state\_s state\_c :=

$$\forall(l: \text{loc}), \text{list\_forall2} \leq_{\text{def,byte}} (\text{get\_bytes } l \text{ state}_s) \\ (\text{get\_bytes } l \text{ state}_c).$$

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

## Another small problem

### Insufficient checks in inj\_value and proj\_value

```
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
```

(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.