Handcrafted Small Inversions Made Operational on Operational Semantics

Jean-François Monin & Xiaomu Shi

LIAMA Beijing, VERIMAG Grenoble

Outline

Inversion

Forward reasoning step of choice

Given a goal containing an hypothesis $H : P$
such that P is (co)inductively defined
extract all useful information from H in the environment

Special case of Case Analysis

But a subtle one : naïve use of `case` or `destruct` fails

Simple example

```
Inductive te : Type :=
  | te_const : nat -> te
  | te_plus : te -> te -> te
  | te_div0 : te -> te.

Inductive val : Type :=
  | nval  : nat -> val
  | bval  : bool -> val.

Inductive eval : te -> val -> Prop :=
  | E_Const : forall n,
      eval (te_const n) (nval n)
  | E_Plus : forall t1 t2 n1 n2,
      eval t1 (nval n1) ->
      eval t2 (nval n2) ->
      eval (te_plus t1 t2) (nval (plus n1 n2)).
```

Coq built-in inversion (Cornes & Terrasse, 1995)

```
Inductive eval : te -> val -> Prop :=
  | E_Const : forall n,
      eval (te_const n) (nval n)
  | E_Plus : forall t1 t2 n1 n2,
      eval t1 (nval n1) ->
      eval t2 (nval n2) ->
      eval (te_plus t1 t2) (nval (plus n1 n2)).
```

Consider the goal:

```
e : eval (te_div0 (te_const 1)) v
============================
3 = 5
```

Using Coq built-in inversion:

```
inversion e.
```


Coq built-in inversion (Cornes & Terrasse, 1995)

```
Inductive eval : te -> val -> Prop :=
  | E_Const : forall n,
      eval (te_const n) (nval n)
  | E_Plus : forall t1 t2 n1 n2,
      eval t1 (nval n1) ->
      eval t2 (nval n2) ->
      eval (te_plus t1 t2) (nval (plus n1 n2)).
```

Consider the goal:

```
e : eval (te_plus (te_const 1) (te_const 2)) v
============================
v = nval 3
```


Coq built-in inversion (Cornes & Terrasse, 1995)

```
e : eval (te_plus (te_const 1) (te_const 2)) v
============================
v = nval 3
```

Using Coq built-in inversion:

```
inversion e; subst.
...
H1 : eval (te_const 1) (nval n1)
H3 : eval (te_const 2) (nval n2)
============================
nval (n1 + n2) = nval 3
```


Issue when using CompCert C semantics

```
H:eval_expr (Genv.globalenv prog_adc) e m RV
   (Ecall (Evalof (Evar copy_StatusRegister T14) T14)
      (Econs
         (Eaddrof
            (Efield (Ederef (Evalof (Evar proc T3) T3) T6)
              adc_compcert.cpsr T7) T8)
         (Econs
            (Ecall (Evalof (Evar spsr T15) T15)
               (Econs (Evalof (Evar proc T3) T3) Enil) T8)
             Enil))
      T12) t m' a'
============================
   proc_state_related m' e st'
```

```
inv H. inv H4. inv H9. inv H5. inv H4. inv H5.
inv H15. inv H4. inv H5. inv H14. inv H4. inv H3.
inv H15. inv H5. inv H4. inv H5. inv H21. inv H13.
...
```


Summary on built-in inversion

Nice features

- Automatic
- Works in most practical cases

Drawbacks

- Basic version: no control on generated names, scripts are unmanageable during maintenance.
- Controlled version `inversion...as...` : have to manually give a lot of names.
- The underlying proof term is complicated → time consuming when designing the script.
- Especially harmful with large inductive types, e.g. from Compcert

Absurd Cases (Small Inversions, 2010)

```
e : eval (te_div0 (te_const 1)) v
============================
3 = 5
```

```
pose (diag t :=
match t with
  | te_div0 (te_const 1) => 3 = 5
  | _ => True
end).
change (diag (te_div0 (te_const 1))).
destruct e; simpl; exact I.
```


Small Inversions, 2010

- Depends on
  - the hypothesis to be inverted (OK)
  - the conclusion (questionable) unconvenient for massive usage
- Issues when considering relevant cases
  - additional interactions
  - not satisfactory for inversions leading to further interactions
- But the approach is very flexible

A more modular variant

```
Definition inv_eval_1_div0 t v (e: eval t v) :=
  let diag t :=
  match t with
    | te_div0 n => ∀ X: Prop, X
    | _ => True
  end
  in match e in eval t v return diag t with
       | E_Const n => I
       | E_Plus _ _ n1 n2 H1 H2 => I
  end.

e : eval (te_div0 (te_const 1)) v
============================
3 = 5

  apply (inv_eval_1_div0 e).
```


The diagonalization function

- yields the premises of focused constructor
- independent from specific conclusion
- takes bindings into account

For constructor `E_Plus`:

```
diag t v := match t with
  | te_plus tc1 tc2 =>
      ∀ X: te -> Prop,
        (∀ n1 n2, eval tc1 (nval n1) ->
                  eval tc2 (nval n2) ->
                  X (nval (plus n1 n2))) -> X v
  | _ => True
end
```


Inversion function for `E_plus`

```
Definition inv_plus_1 {t} {v} (e: eval t v) :=
  match e in (eval t v) return diag t v with
    | E_Plus _ _ n1 n2 e1 e2 => (fun X k => k n1 n2 e1 e2)
    | _ => I
  end.
```

Usage:

```
e : eval (te_plus (te_const 1) (te_const 2)) v
============================
v = nval 3

apply (inv_plus_1 e); intros n1 n2 e1 e2.

e1 : eval (te_const 1) (nval n1)
e2 : eval (te_const 2) (nval n2)
============================
nval (n1 + n2) = nval 3
```


More on inversion functions

Variant are possible, e.g.
merging inversion functions dedicated to all constructors

Different variants are needed in different situations

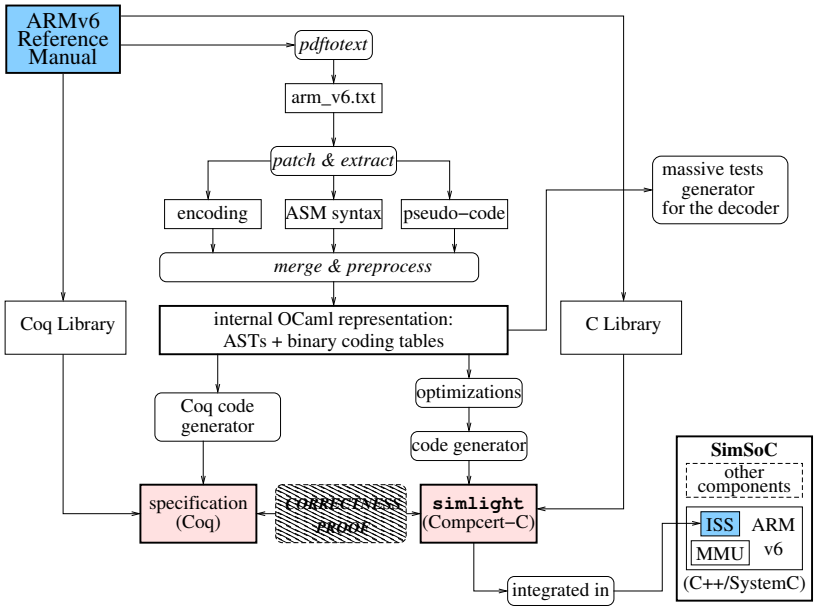
See demo

Handcrafted inversion may beat standard inversion

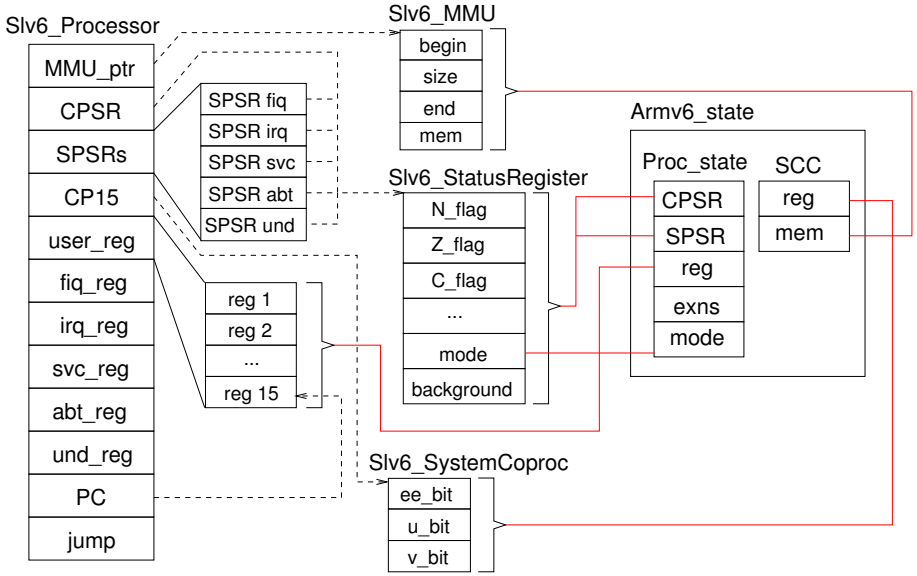
```
Inductive t : nat -> Set :=
    | F1 : ∀ {n}, t (S n)
    | FS : ∀ {n}, t n -> t (S n).

Inductive odd : ∀ n : nat, t n -> Prop :=
   | odd_1 : ∀ n, odd (S n) F1
   | odd_SS : ∀ n i, odd n i -> odd _ (FS (FS i)).
```

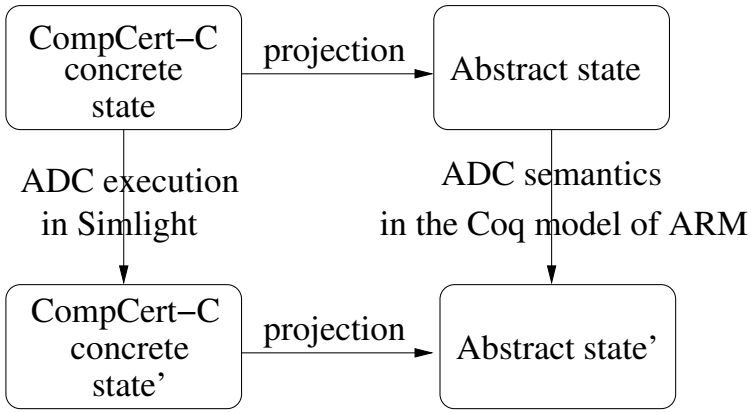

SimSoC: a simulator of systems-on-chip



Projection of ARMv6 processor state (excerpt)



Main theorem for ADC (add with carry)



Evaluation rule of a field in CompCert C semantics

```
Inductive eval_expr :
 env-> mem-> kind-> expr-> trace-> mem-> expr-> Prop:=
 ...
 | eval_field : ∀ e m a t m' a' f ty,
    eval_expr e m RV a t m' a' ->
    eval_expr e m LV (Efield a f ty) t m' (Efield a' f ty)
```


Inversion function for `eval_field`

```
Definition inv_field g e m ex t m' ex'
  (ee:eval_expr g e m LV ex t m' ex') :=
  let diag e ex ex' m m' :=
    match ex with
      | Efield a b c =>
        ∀ (X:expr->Prop),
         (∀ t a', eval_expr g e m RV a t m' a' ->
                              X (Efield a' b c)) -> X ex'
      | _ => True
    end in
    match ee in (eval_expr _ e m _ ex _ m' ex')
             return  diag e ex ex' m m'  with
      | eval_field _ _ _ t _ a' _ _ H1 =>
          fun X k => k t a' H1
      | _ => I
    end.
```


High-level tactic for inductive type `eval_expr`

`inv_eval_expr`: repeat the following steps

- find the hypothesis we want to invert
  (by matching the targetted memory states);
- revert related hypotheses;
- call the right diagonal function;
- give meaningful names to derived variables and hypotheses
  (predictable from `inv_field`, etc.);
- update all other related hypotheses;
- clean up useless variables and hypotheses;

High-level tactic for inductive type `eval_expr`

```
Ltac inv_eval_expr m m' :=
  ...
  let t1_:=fresh "t" in
  let v1_:=fresh "v" in
  let ev_ex1 := fresh "ev_ex" in
  ...
  match goal with
  ...
  |[ee: eval_expr ?ge ?e m LV (Efield ?a ?f ?ty) ?t m' ?a'
      |- ?cl] =>
      apply (inv_field ee);
      clear ee; intros t1_ a1_ ev_ex1;
      intros;
      inv_eval_expr m m'
```

Built-in inversion

```
H:eval_expr (Genv.globalenv prog_adc) e m RV
   (Ecall (Evalof (Evar copy_StatusRegister T14) T14)
      (Econs
         (Eaddrof
            (Efield (Ederef (Evalof (Evar proc T3) T3) T6)
              adc_compcert.cpsr T7) T8)
         (Econs
            (Ecall (Evalof (Evar spsr T15) T15)
               (Econs (Evalof (Evar proc T3) T3) Enil) T8)
             Enil))
      T12) t m' a'
============================
   proc_state_related m' e st'

  inv H. inv H4. inv H9. inv H5. inv H4. inv H5.
  inv H15. inv H4. inv H5. inv H14. inv H4. inv H3.
  inv H15. inv H5. inv H4. inv H5. inv H21. inv H13.
  ...
```


Handcrafted small inversion

```
H:eval_expr (Genv.globalenv prog_adc) e m RV
   (Ecall (Evalof (Evar copy_StatusRegister T14) T14)
      (Econs
         (Eaddrof
            (Efield (Ederef (Evalof (Evar proc T3) T3) T6)
              adc_compcert.cpsr T7) T8)
         (Econs
            (Ecall (Evalof (Evar spsr T15) T15)
               (Econs (Evalof (Evar proc T3) T3) Enil) T8)
             Enil))
      T12) t m' a'
============================
   proc_state_related m' e st'

  inv_eval_expr m m'.
  ...
```


Performance

Table: Time costs (in seconds)

	inversion	D. Inversion	BasicElim	hc_inversion
Full ex.	1.628	0.976	1.428	0.312
Ecall	0.132	0.076	0.112	0.028
Evalof	0.132	0.072	0.092	0.020
Evar	0.128	0.064	0.084	0.024
Eaddrof	0.140	0.076	0.104	0.020

Table: Size of compilation results (in KBytes)

	inversion	D. Inversion	BasicElim	hc_inversion
Full ex.	191	460	171	37

D. Inversion = Derived Inversion

Conclusion

Yes you can

do inversions by yourself

Thanks!

Questions?