Implementing `capture` on CompCert

JUNEYOUNG LEE

MAY. 14

Defining `capture` function

Commit :

https://github.com/aqjune/CompCert-intptr/commit/3d7eeb35825b706c246376fa3a2f00086f7fbe43

What is `capture`?

C

```
char *ptr = malloc();
int i = (int)ptr;
```

Low-level lang.

```
char *ptr = malloc();
capture(ptr); // lowers
int i = (int)ptr;
```

Development

GitHub repo :

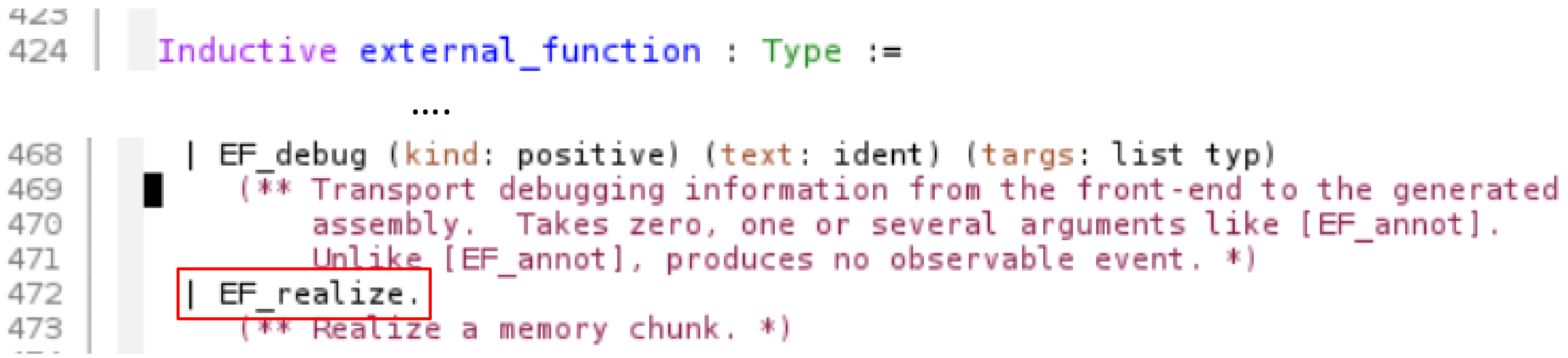
https://github.com/aqjune/CompCert-intptr/tree/add_realize

Current Status :

- “capture” external function added (done)
- Inserting capture when lowering C is on progress
- Proof is on progress (some Qed, some admit/Admitted)

Defining `capture` External Function

“EF_realize”

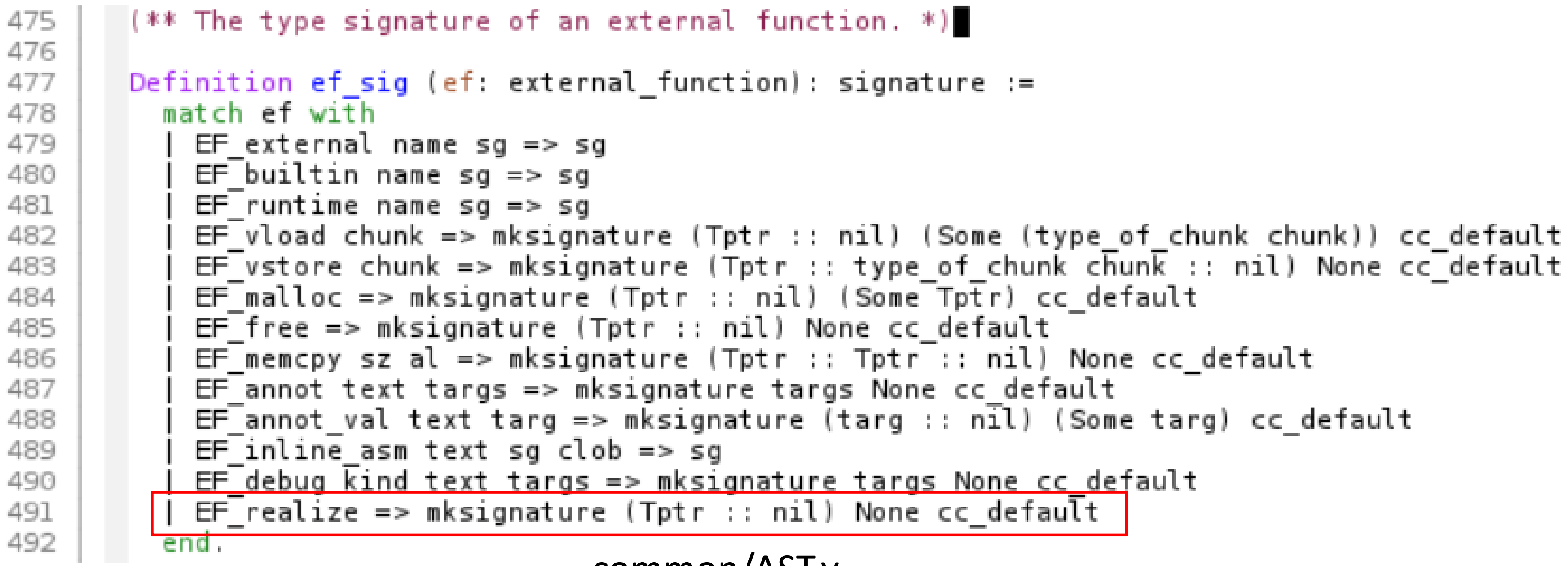


```
Inductive external_function : Type :=
                 ....
  | EF_debug (kind: positive) (text: ident) (targs: list typ)
     (** Transport debugging information from the front-end to the generated
         assembly.  Takes zero, one or several arguments like [EF_annot].
         Unlike [EF_annot], produces no observable event. *)
  | EF_realize.
     (** Realize a memory chunk. *)
```

common/AST.v

(`realize` = past name of `capture`)

Defining `capture` External Function



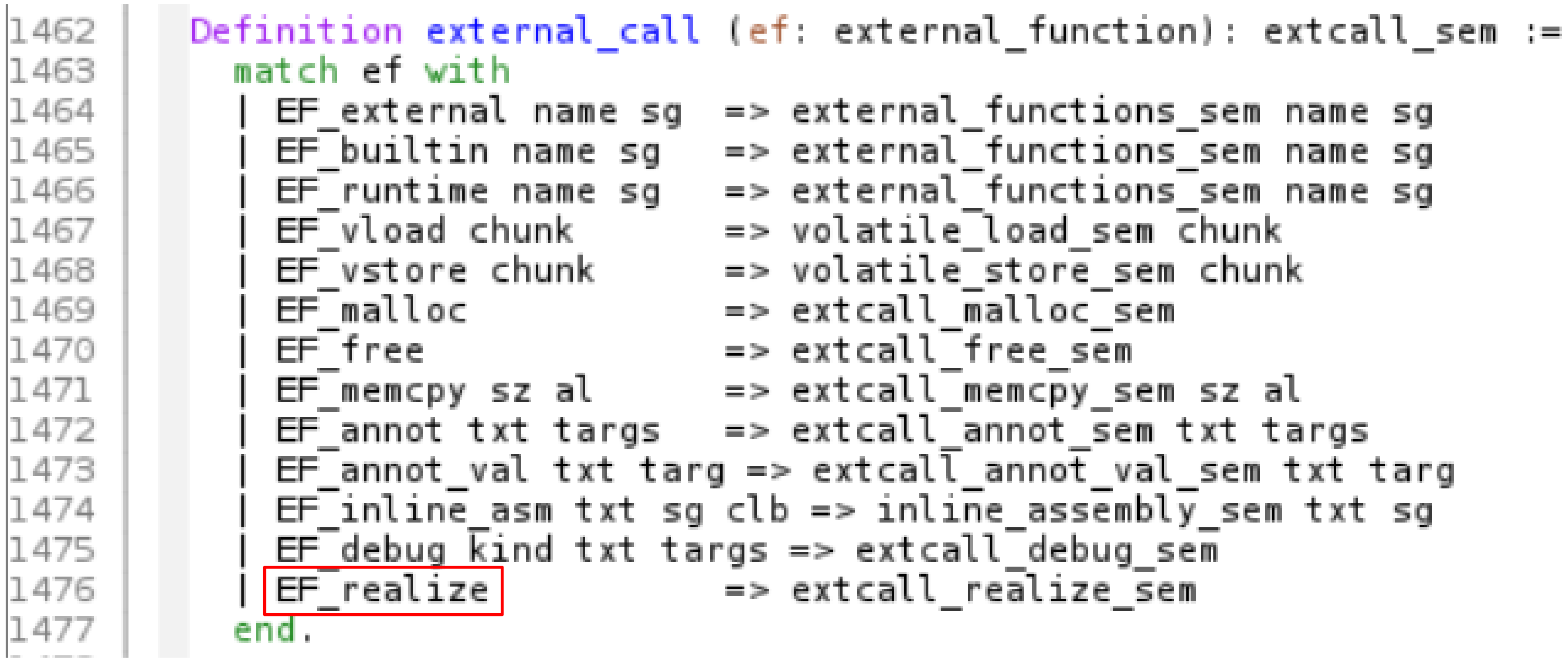
```
(** The type signature of an external function. *)

Definition ef_sig (ef: external_function): signature :=
  match ef with
  | EF_external name sg => sg
  | EF_builtin name sg => sg
  | EF_runtime name sg => sg
  | EF_vload chunk => mksignature (Tptr :: nil) (Some (type_of_chunk chunk)) cc_default
  | EF_vstore chunk => mksignature (Tptr :: type_of_chunk chunk :: nil) None cc_default
  | EF_malloc => mksignature (Tptr :: nil) (Some Tptr) cc_default
  | EF_free => mksignature (Tptr :: nil) None cc_default
  | EF_memcpy sz al => mksignature (Tptr :: Tptr :: nil) None cc_default
  | EF_annot text targs => mksignature targs None cc_default
  | EF_annot_val text targ => mksignature (targ :: nil) (Some targ) cc_default
  | EF_inline_asm text sg clob => sg
  | EF_debug kind text targs => mksignature targs None cc_default
  | EF_realize => mksignature (Tptr :: nil) None cc_default
  end.
```

common/AST.v

+ some aspects of the ‘realize’ function (ex : will the function be inlined by compiler?)

Semantics of `capture`

Currently defined as no-op.

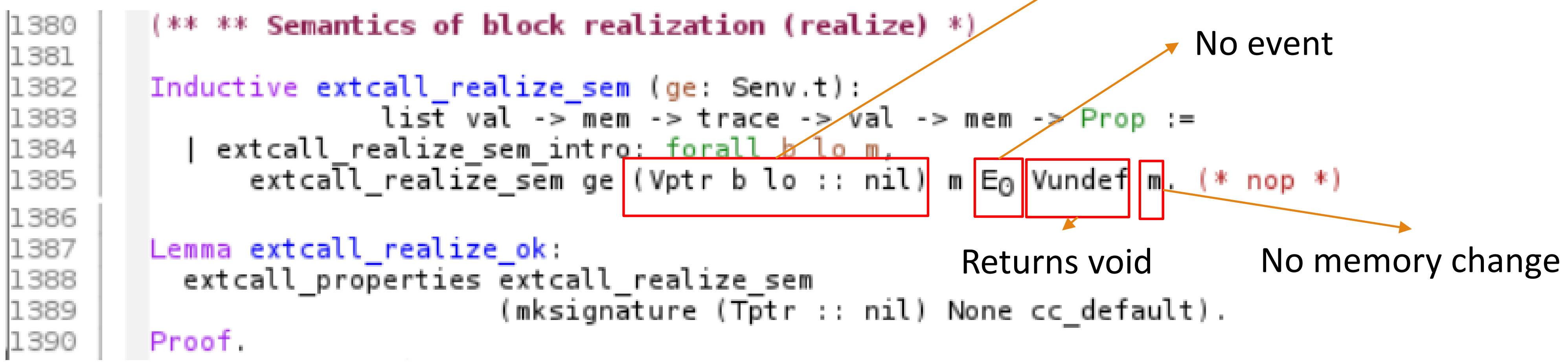


```
Definition external_call (ef: external_function): extcall_sem :=
  match ef with
  | EF_external name sg  => external_functions_sem name sg
  | EF_builtin name sg   => external_functions_sem name sg
  | EF_runtime name sg   => external_functions_sem name sg
  | EF_vload chunk       => volatile_load_sem chunk
  | EF_vstore chunk      => volatile_store_sem chunk
  | EF_malloc            => extcall_malloc_sem
  | EF_free              => extcall_free_sem
  | EF_memcpy sz al      => extcall_memcpy_sem sz al
  | EF_annot txt targs   => extcall_annot_sem txt targs
  | EF_annot_val txt targ => extcall_annot_val_sem txt targ
  | EF_inline_asm txt sg clb => inline_assembly_sem txt sg
  | EF_debug kind txt targs => extcall_debug_sem
  | EF_realize           => extcall_realize_sem
  end.
```

common/Events.v

Semantics of `capture`

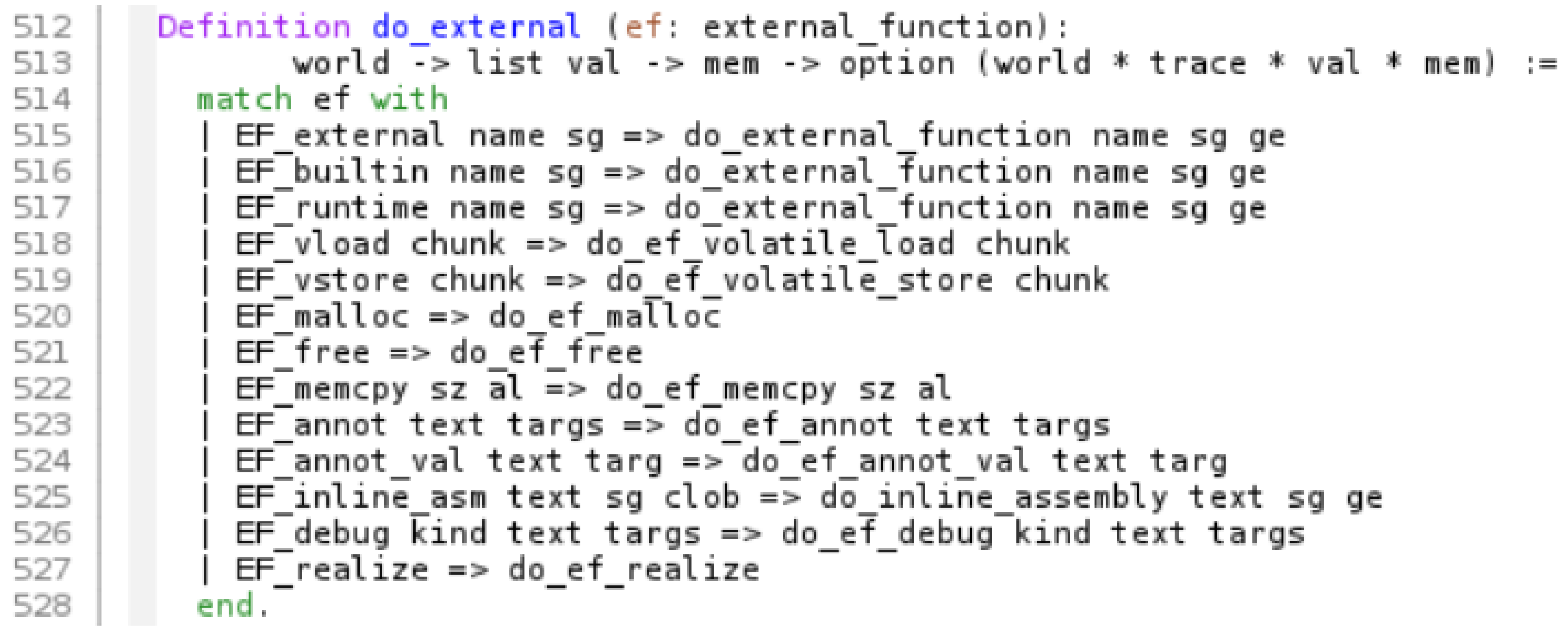
Currently defined as no-op.



```
(** ** Semantics of block realization (realize) *)

Inductive extcall_realize_sem (ge: Senv.t):
            list val -> mem -> trace -> val -> mem -> Prop :=
  | extcall_realize_sem_intro: forall b lo m,
      extcall_realize_sem ge (Vptr b lo :: nil) m E0 Vundef m. (* nop *)

Lemma extcall_realize_ok:
  extcall_properties extcall_realize_sem
                     (mksignature (Tptr :: nil) None cc_default).
Proof.
```

common/Events.v

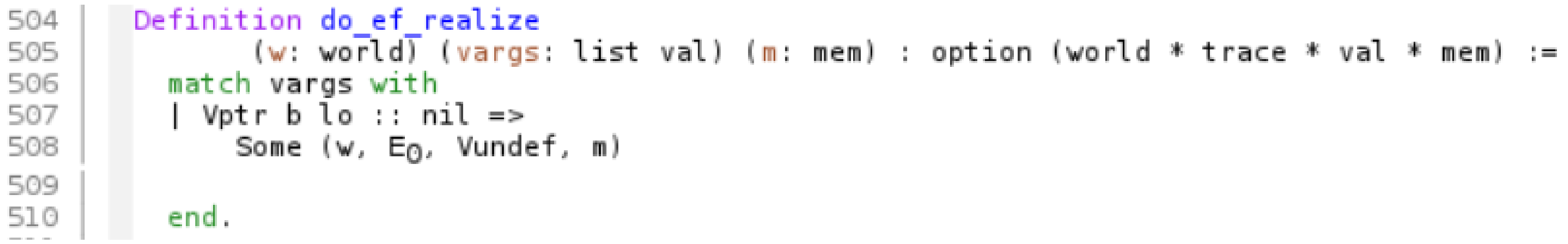
Execution of `capture`



```
Definition do_external (ef: external_function):
       world -> list val -> mem -> option (world * trace * val * mem) :=
  match ef with
  | EF_external name sg => do_external_function name sg ge
  | EF_builtin name sg => do_external_function name sg ge
  | EF_runtime name sg => do_external_function name sg ge
  | EF_vload chunk => do_ef_volatile_load chunk
  | EF_vstore chunk => do_ef_volatile_store chunk
  | EF_malloc => do_ef_malloc
  | EF_free => do_ef_free
  | EF_memcpy sz al => do_ef_memcpy sz al
  | EF_annot text targs => do_ef_annot text targs
  | EF_annot_val text targ => do_ef_annot_val text targ
  | EF_inline_asm text sg clob => do_inline_assembly text sg ge
  | EF_debug kind text targs => do_ef_debug kind text targs
  | EF_realize => do_ef_realize
  end.
```

cfrontend/Cexec.v

Execution of `capture`



```
Definition do_ef_realize
        (w: world) (vargs: list val) (m: mem) : option (world * trace * val * mem) :=
  match vargs with
  | Vptr b lo :: nil =>
      Some (w, E0, Vundef, m)

  end.
```

cfrontend/Cexec.v

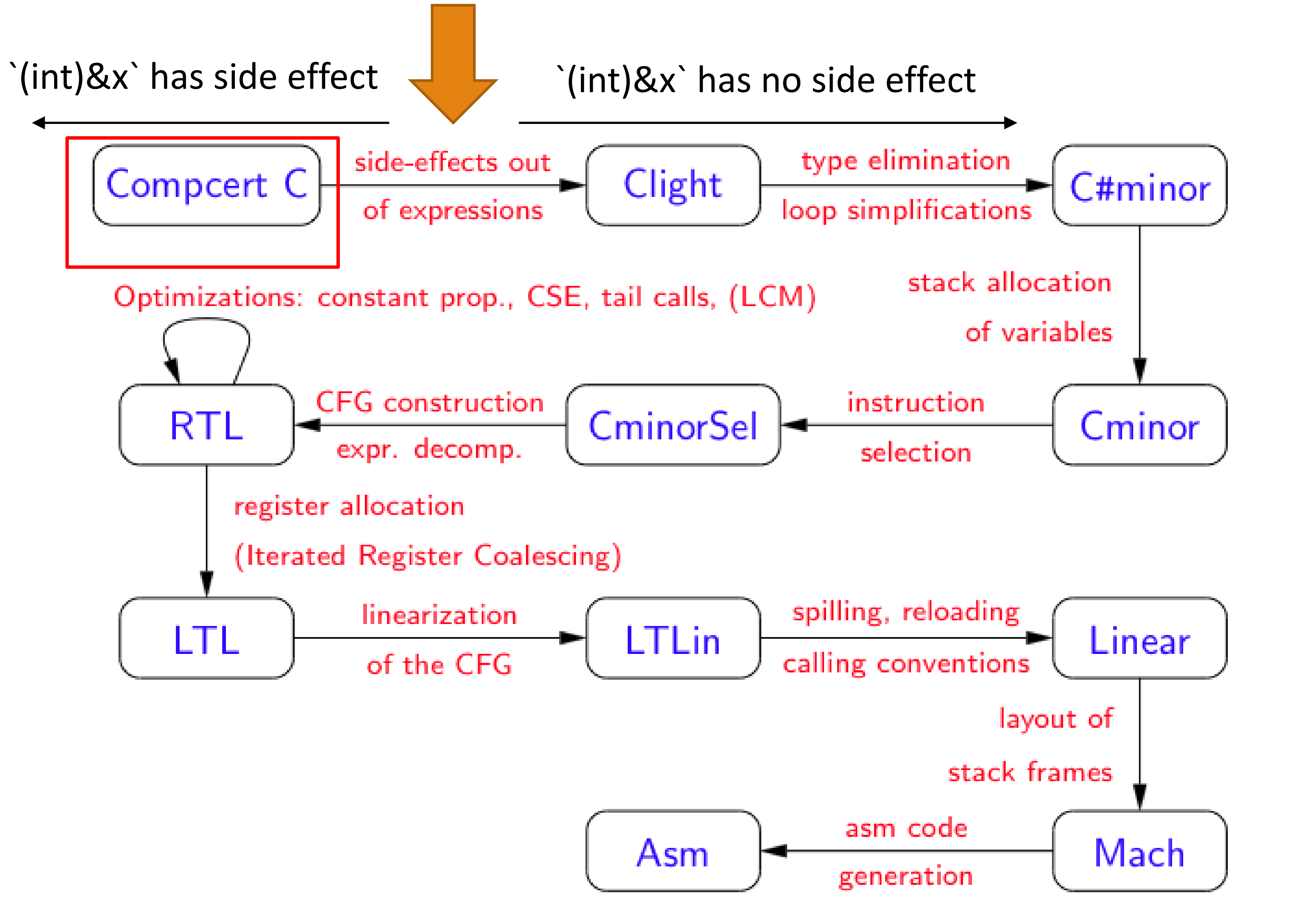
Modifying CompCert C semantics

Commit :

https://github.com/aqjune/CompCert-intptr/commit/2882abcdcf20a6449c424ad4a74311bf41a00a31

Languages in CompCert

Insert `capture` whenever pointer -> integer casting happens!



`(int)&x` has side effect
`(int)&x` has no side effect
Compcert C
side-effects out of expressions
Clight
type elimination loop simplifications
C#minor
stack allocation of variables
Cminor
instruction selection
CminorSel
CFG construction expr. decomp.
RTL
Optimizations: constant prop., CSE, tail calls, (LCM)
register allocation (Iterated Register Coalescing)
LTL
linearization of the CFG
LTLin
spilling, reloading calling conventions
Linear
layout of stack frames
Mach
asm code generation
Asm

Reduction Semantics of CompCert C

cfrontend/Cexec.v

```
Fixpoint step_expr (k: kind) (a: expr) (m: mem): reducts expr :=
```

- Def. of `kind` :

```
Inductive kind : Type := LV | RV.
```
(cfrontend/Csem.v)

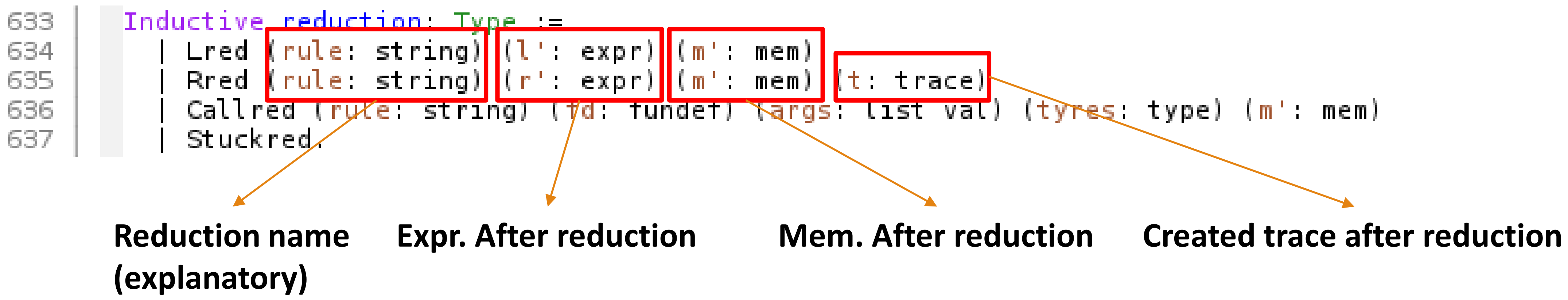
- Def. of `expr` : cfrontend/Csyntax.v

- Def. of `mem` : common/Memory.v

- Def. of `reducts` : (next slide)

Reduction Semantics of CompCert C

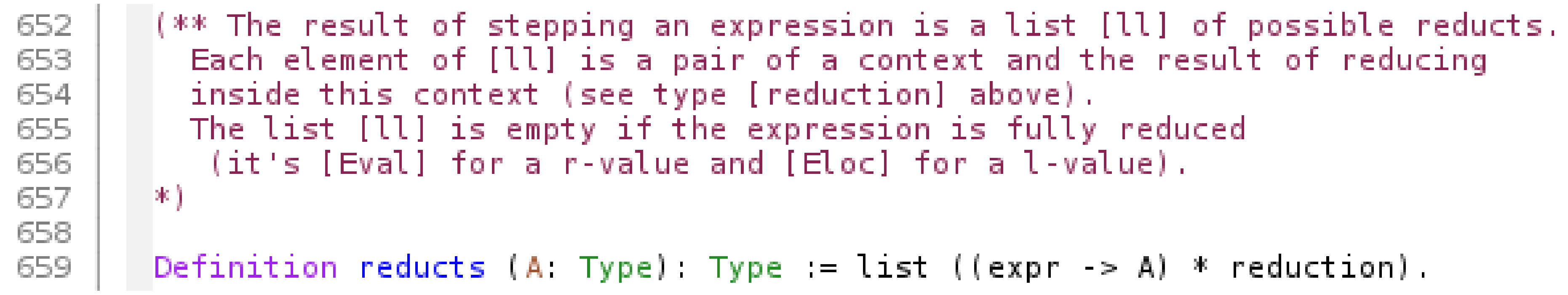
cfrontend/Cexec.v



```
Inductive reduction: Type :=
  | Lred (rule: string) (l': expr) (m': mem)
  | Rred (rule: string) (r': expr) (m': mem) (t: trace)
  | Callred (rule: string) (fd: fundef) (args: list val) (tyres: type) (m': mem)
  | Stuckred.
```

Reduction Semantics of CompCert C

cfrontend/Cexec.v



```
(** The result of stepping an expression is a list [ll] of possible reducts.
  Each element of [ll] is a pair of a context and the result of reducing
  inside this context (see type [reduction] above).
  The list [ll] is empty if the expression is fully reduced
   (it's [Eval] for a r-value and [Eloc] for a l-value).
*)

Definition reducts (A: Type): Type := list ((expr -> A) * reduction).
```

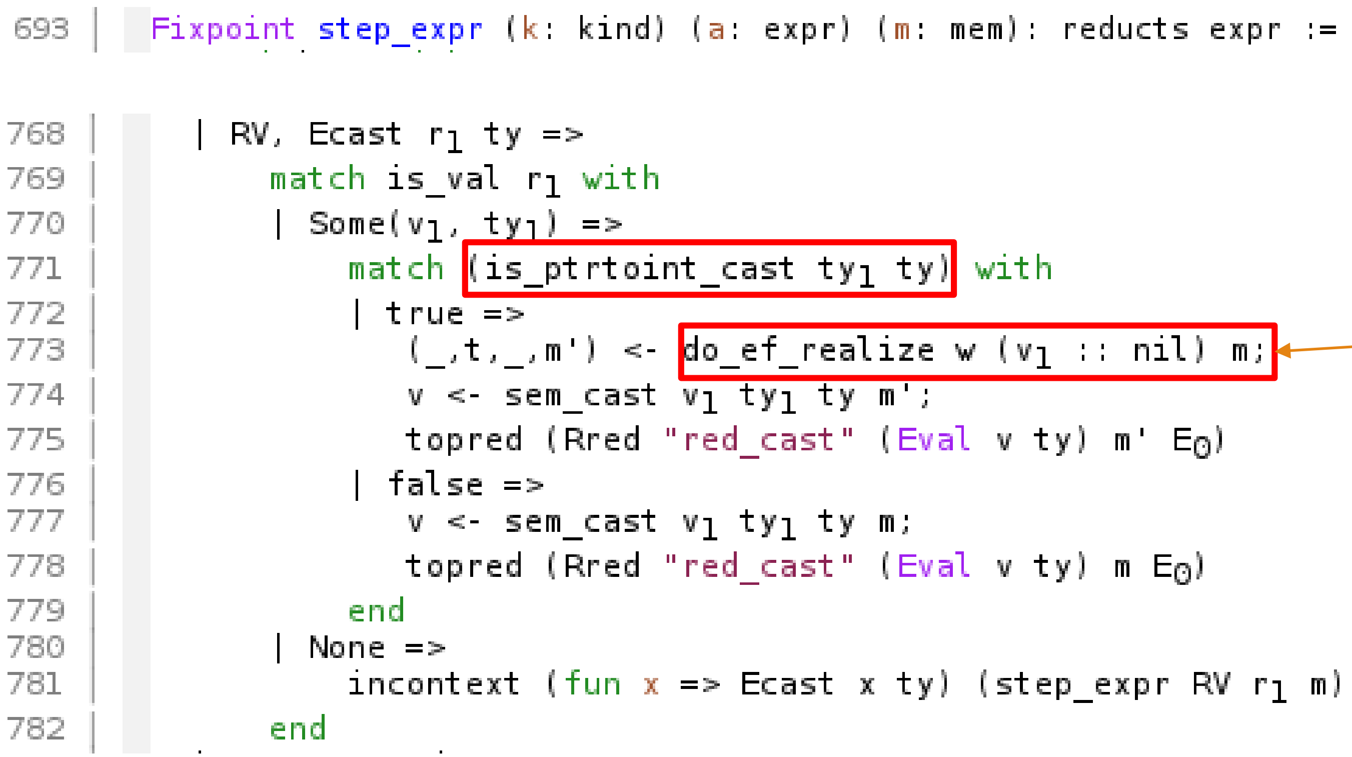
A : either expr or exprlist

(expr -> A) part : context
- Applying “expr” field of the second term to the context makes a full expr.

Why list ? ➔ C is Nondeterministic! (ex : `*f(&x) = 1 + 2`)

Ptr2Int Casting in CompCert C

cfrontend/Cexec.v

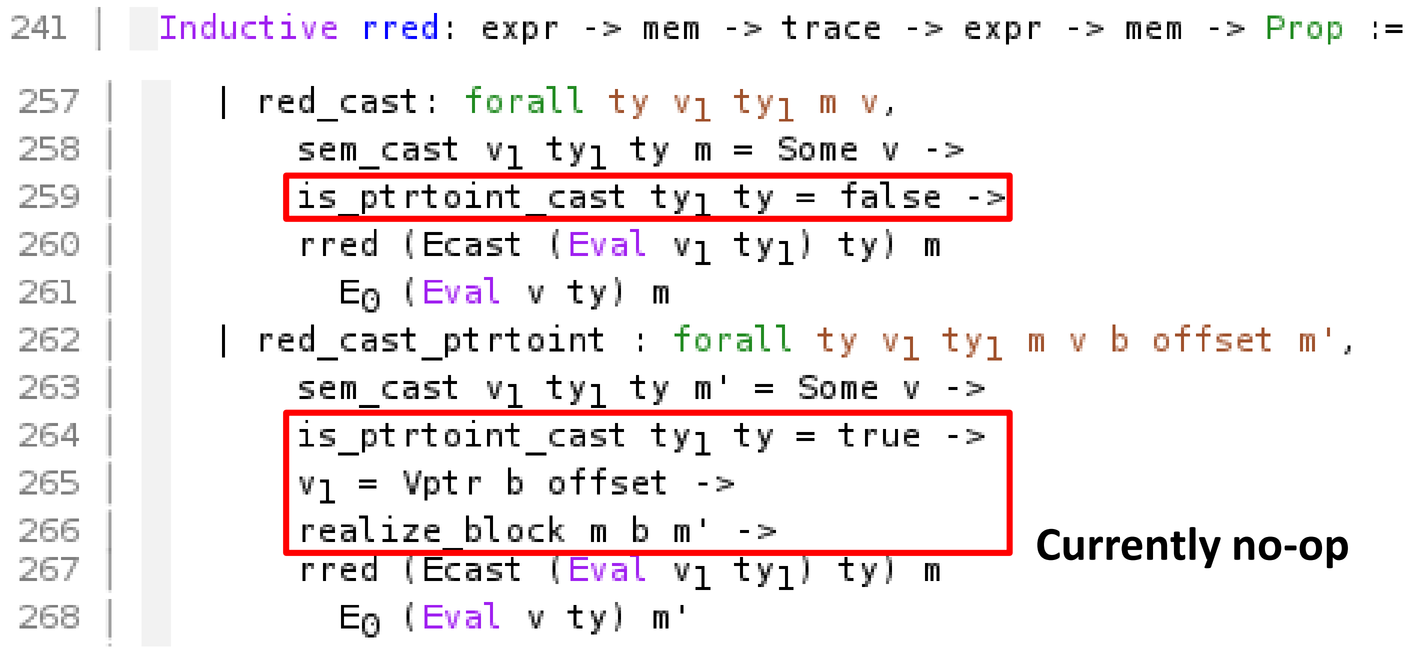


```
Fixpoint step_expr (k: kind) (a: expr) (m: mem): reducts expr :=

  | RV, Ecast r1 ty =>
      match is_val r1 with
      | Some(v1, ty1) =>
        match (is_ptrtoint_cast ty1 ty) with
        | true =>
          (_,t,_,m') <- do_ef_realize w (v1 :: nil) m;
          v <- sem_cast v1 ty1 ty m';
          topred (Rred "red_cast" (Eval v ty) m' E0)
        | false =>
          v <- sem_cast v1 ty1 ty m;
          topred (Rred "red_cast" (Eval v ty) m E0)
        end
      | None =>
        incontext (fun x => Ecast x ty) (step_expr RV r1 m)
      end
```

Currently no-op

Ptr2Int Casting in CompCert C (Sem.)

cfrontend/Csem.v



```
Inductive rred: expr -> mem -> trace -> expr -> mem -> Prop :=

    | red_cast: forall ty v1 ty1 m v,
        sem_cast v1 ty1 ty m = Some v ->
        is_ptrtoint_cast ty1 ty = false ->
        rred (Ecast (Eval v1 ty1) ty) m
          E0 (Eval v ty) m
    | red_cast_ptrtoint : forall ty v1 ty1 m v b offset m',
        sem_cast v1 ty1 ty m' = Some v ->
        is_ptrtoint_cast ty1 ty = true ->
        v1 = Vptr b offset ->
        realize_block m b m' ->
        rred (Ecast (Eval v1 ty1) ty) m
          E0 (Eval v ty) m'
```

Currently no-op

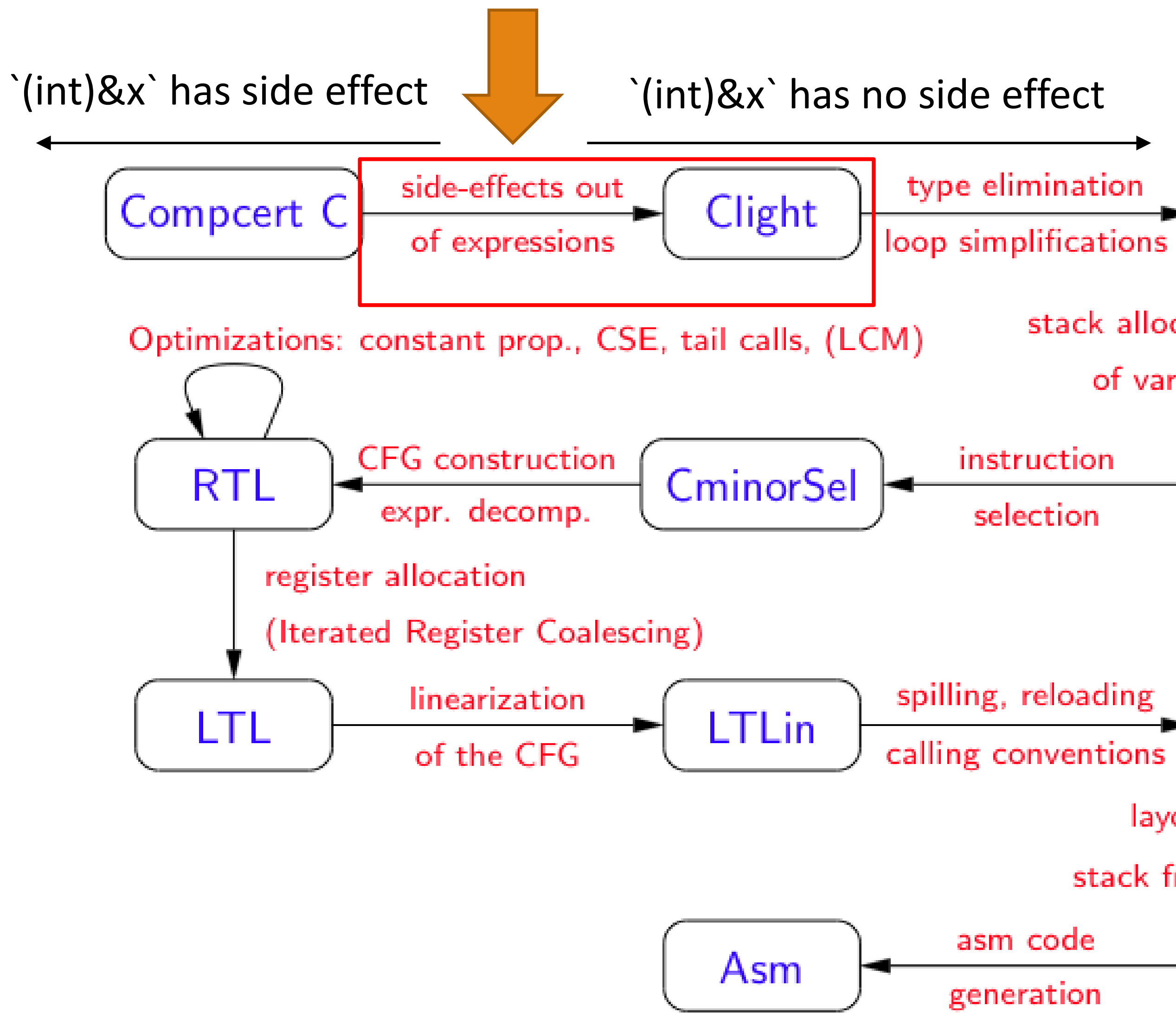
Lowering CompCert C to Clight

Commit :

https://github.com/aqjune/CompCert-intptr/commit/2882abcdcf20a6449c424ad4a74311bf41a00a31

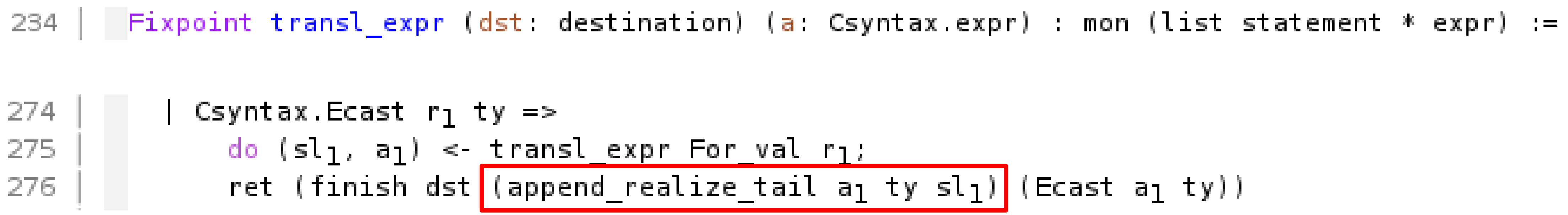
Languages in CompCert

Insert `capture` whenever pointer -> integer casting happens!


`(int)&x` has side effect
`(int)&x` has no side effect
Compcert C
side-effects out of expressions
Clight
type elimination loop simplifications
C#minor
stack allocation of variables
Cminor
instruction selection
CminorSel
CFG construction expr. decomp.
RTL
Optimizations: constant prop., CSE, tail calls, (LCM)
register allocation (Iterated Register Coalescing)
LTL
linearization of the CFG
LTLin
spilling, reloading calling conventions
Linear
layout of stack frames
Mach
asm code generation
Asm

CompCert C → Clight

cfrontend/SimplExpr.v



```
Fixpoint transl_expr (dst: destination) (a: Csyntax.expr) : mon (list statement * expr) :=

  | Csyntax.Ecast r1 ty =>
      do (sl1, a1) <- transl_expr For_val r1;
      ret (finish dst (append_realize_tail a1 ty sl1) (Ecast a1 ty))
```

Works to do

Works to do

1. Implicit Casts

- There are expressions that do implicit cast in CompCert C

- These should be covered

2. Proof

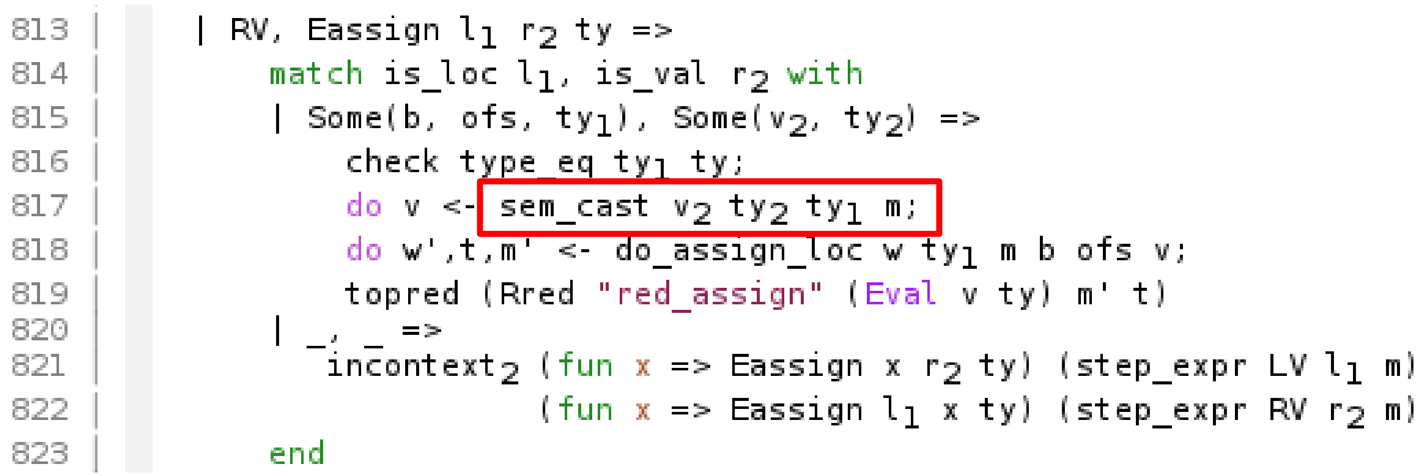
- Currently the most problematic point : Proof that uses ‘side-effect-lessness’ of casting

3. Deploy int-ptr memory model

- Lots and lots of work..

Ptr2Int Casting– Works To Do 1

(Implicit ptr -> int casting ; should be fixed!)

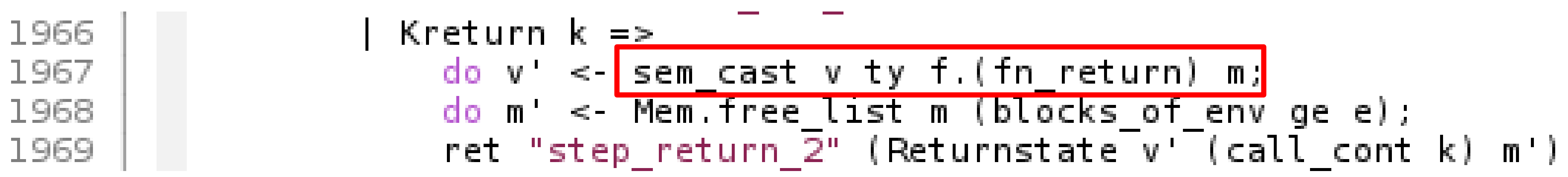


```
| RV, Eassign l1 r2 ty =>
    match is_loc l1, is_val r2 with
    | Some(b, ofs, ty1), Some(v2, ty2) =>
        check type_eq ty1 ty;
        do v <- sem_cast v2 ty2 ty1 m;
        do w',t,m' <- do_assign_loc w ty1 m b ofs v;
        topred (Rred "red_assign" (Eval v ty) m' t)
    | _, _ =>
       incontext2 (fun x => Eassign x r2 ty) (step_expr LV l1 m)
                  (fun x => Eassign l1 x ty) (step_expr RV r2 m)
    end
```

cfrontend/Cexec.v

Ptr2Int Casting– Works To Do 2

(Implicit ptr -> int casting when returning a value (at the end of a function) ; should be fixed!)



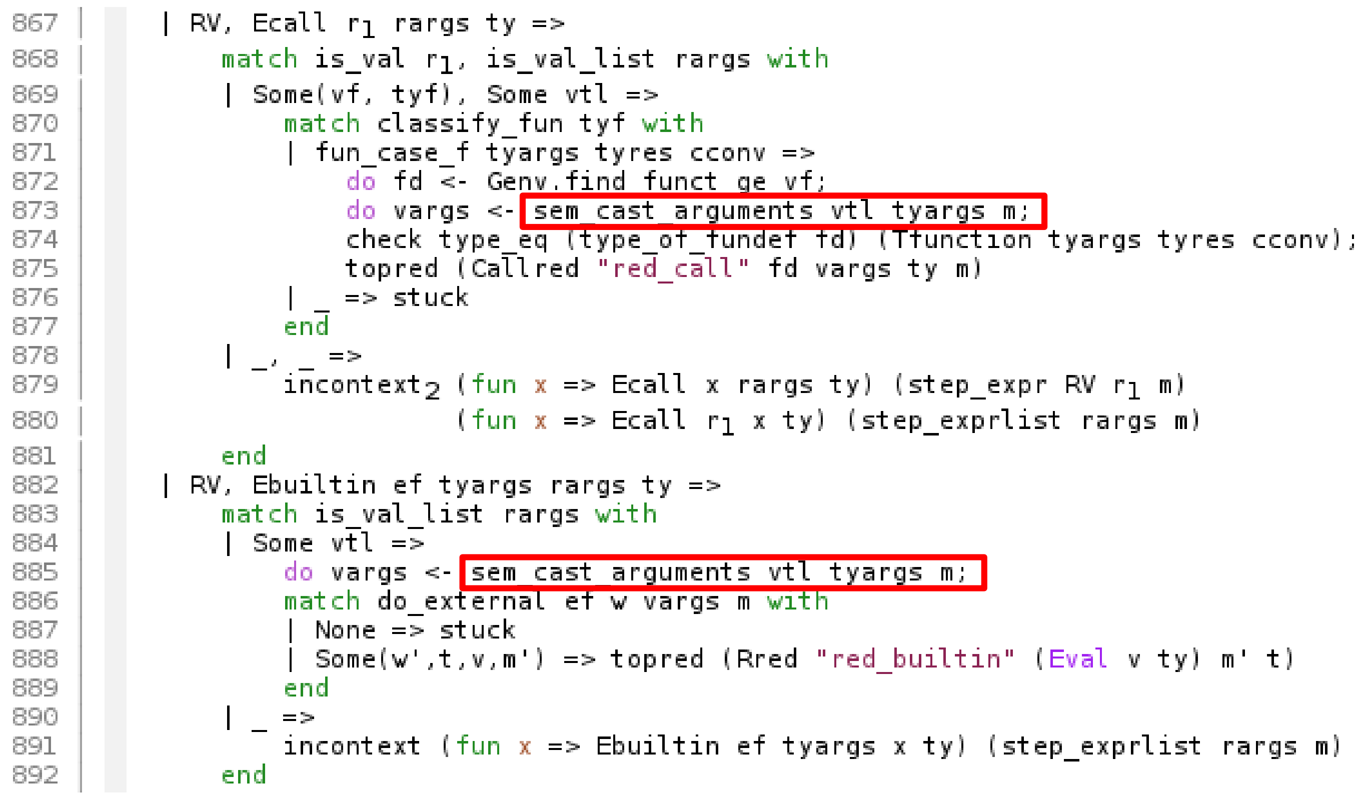
```
      | Kreturn k =>
        do v' <- sem_cast v ty f.(fn_return) m;
        do m' <- Mem.free_list m (blocks_of_env ge e);
        ret "step_return_2" (Returnstate v' (call_cont k) m')
```

cfrontend/Cexec.v

Ptr2Int Casting– Works To Do 3

(Implicit ptr -> int casting when calling a function ; should be fixed!)

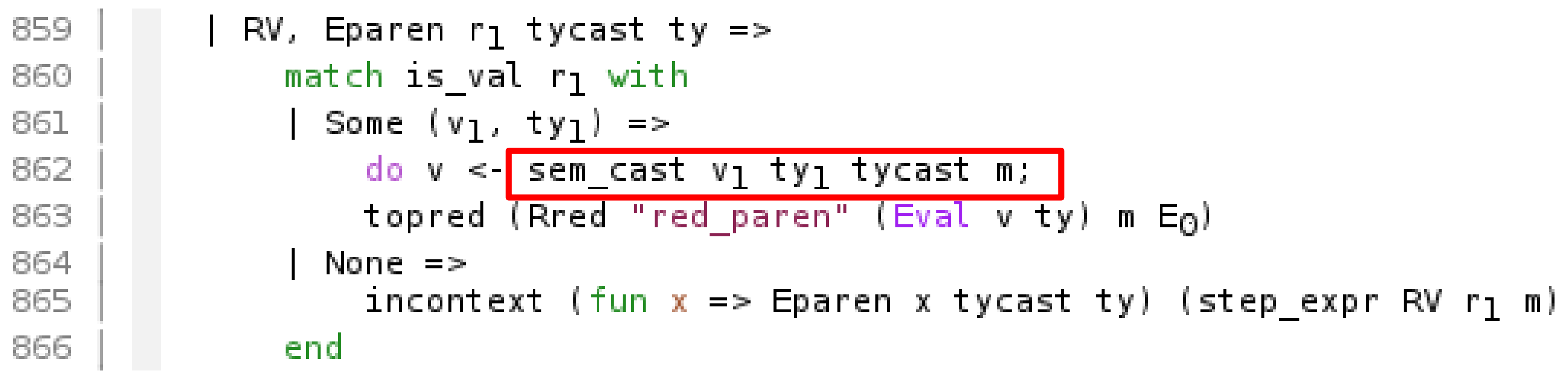
cfrontend/Cexec.v



```
| RV, Ecall r1 rargs ty =>
    match is_val r1, is_val_list rargs with
    | Some(vf, tyf), Some vtl =>
        match classify_fun tyf with
        | fun_case_f tyargs tyres cconv =>
            do fd <- Genv.find_funct ge vf;
            do vargs <- sem_cast_arguments vtl tyargs m;
            check type_eq (type_of_fundef fd) (Tfunction tyargs tyres cconv);
            topred (Callred "red_call" fd vargs ty m)
        | _ => stuck
        end
    | _, _ =>
        incontext2 (fun x => Ecall x rargs ty) (step_expr RV r1 m)
                   (fun x => Ecall r1 x ty) (step_exprlist rargs m)
    end
| RV, Ebuiltin ef tyargs rargs ty =>
    match is_val_list rargs with
    | Some vtl =>
        do vargs <- sem_cast_arguments vtl tyargs m;
        match do_external ef w vargs m with
        | None => stuck
        | Some(w',t,v,m') => topred (Rred "red_builtin" (Eval v ty) m' t)
        end
    | _ =>
        incontext (fun x => Ebuiltin ef tyargs x ty) (step_exprlist rargs m)
    end
```

Ptr2Int Casting– Works To Do 4

(Parenthesis operator ; I don't know what it exactly is..)



```
| RV, Eparen r1 tycast ty =>
  match is_val r1 with
  | Some (v1, ty1) =>
    do v <- sem_cast v1 ty1 tycast m;
    topred (Rred "red_paren" (Eval v ty) m E0)
  | None =>
    incontext (fun x => Eparen x tycast ty) (step_expr RV r1 m)
  end
```

cfrontend/Cexec.v

Appendix I. extcall_properties

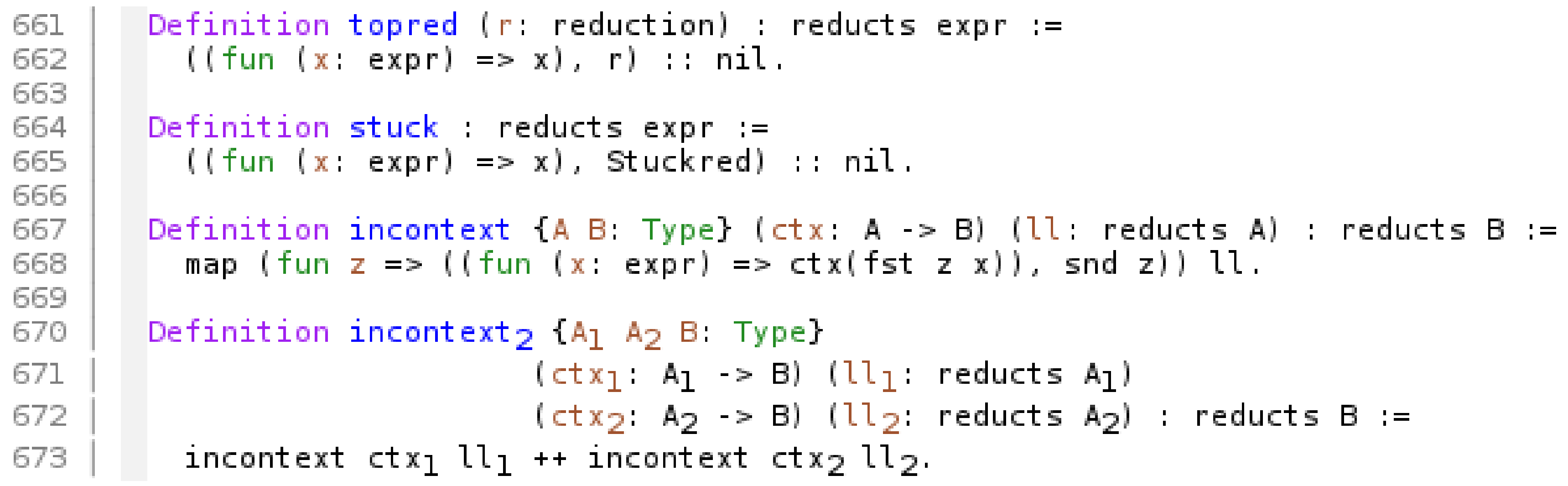
Record extcall_properties (sem: extcall_sem) (sg: signature) : Prop

Common/Events.v

1. The return value of an external call must agree with its signature.
2. External calls cannot invalidate memory blocks. (Remember that freeing a block does not invalidate its block identifier.)
3. External calls cannot increase the max permissions of a valid block.
4. They can decrease the max permissions, e.g. by freeing.
5. External call cannot modify memory unless they have [Max, Writable] permissions.
6. External calls must commute with memory extensions, in the following sense.
7. External calls must commute with memory injections, in the following sense.
8. External calls produce at most one event.
9. External calls must be receptive to changes of traces by another, matching trace.
10. External calls must be deterministic up to matching between traces.

Appendix II. Reduction semantics

Common/Events.v



```
Definition topred (r: reduction) : reducts expr :=
  ((fun (x: expr) => x), r) :: nil.

Definition stuck : reducts expr :=
  ((fun (x: expr) => x), Stuckred) :: nil.

Definition incontext {A B: Type} (ctx: A -> B) (ll: reducts A) : reducts B :=
  map (fun z => ((fun (x: expr) => ctx(fst z x)), snd z)) ll.

Definition incontext2 {A1 A2 B: Type}
                     (ctx1: A1 -> B) (ll1: reducts A1)
                     (ctx2: A2 -> B) (ll2: reducts A2) : reducts B :=
  incontext ctx1 ll1 ++ incontext ctx2 ll2.
```