A Mechanized Theory of Communication Analysis in CML

March 7, 2019

Concurrent ML

- extension of Standard ML
- concurrency and synchronization
- synchronized communication over channels: send event, receive event
- composition of events: choose event, wrap event ...

Concurrent ML

```
type thread_id
val spawn : (unit -> unit) -> thread_id

type 'a chan
val channel : unit -> 'a chan

type 'a event
val sync: 'a event -> 'a

val recvEvt: 'a chan -> 'a event
val sendEvt: 'a channel * 'a -> unit event

val send: 'a chan * 'a -> unit
fun send (ch, v) = sync (sendEvt (ch, v))

val recv: 'a chan -> 'a
fun recv ch = sync (recvEvt ch)
```

Concurrent ML

```
structure Serv : SERV =
struct
  datatype serv =
    S of (int * int chan) chan

  fun make () =
  let
    val reqCh = channel ()
    fun loop state =
    let
      val (v, replCh) = recv reqCh
      val () = send (replCh, state)
    in
      loop v
    end
    val () = spawn (fn () => loop 0)
  in
    S reqCh
  end
```

```
  fun call (server, v) =
  let
    val S reqCh = server
    val replCh = channel ()
    val () = send (reqCh, (v, replCh))
  in
    recv replCh
  end
end

signature SERV =
sig
  type serv
  val make : unit -> serv
  val call : serv * int -> int
end
```

Isabelle/HOL

- interactive theorem proving assistant; proof assistant
- trusted kernel of manipulation rules
- unification and rewriting
- simply typed terms
- propositions as boolean typed terms
- higher order terms
- inductive data
- computable recursive functions
- inductive predicates
- inductive reasoning
- tactics and composition

Isabelle/HOL

```
⊢ P1 ∨ P2 ⟶ Q
proof
  assume P1 ∨ P2:
    case P1:
      have ⊢ P1 ⟶ Q by A
      have ⊢ Q by modus ponens
    case P2:
      have ⊢ P2 ⟶ Q by B
      have ⊢ Q by modus ponens
    have P1 ⊢ Q, P2 ⊢ Q
    have ⊢ Q by disjunction elimination
  have P1 ∨ P2 ⊢ Q
  have ⊢ P1 ∨ P2 ⟶ Q
    by implication introduction
qed
```

```
⊢ P1 ∨ P2 ⟶ Q
apply (rule impI)
  P1 ∨ P2 ⊢ Q
apply (erule disjE)
  P1 ⊢ Q
* P2 ⊢ Q
apply (insert A)
  P1, P1 ⟶ Q ⊢ Q
* P2 ⊢ Q
apply (erule mp)
  P1 ⊢ P1
* P2 ⊢ Q
apply assumption
  P2 ⊢ Q
apply (insert B)
  P2, P2 ⟶ Q ⊢ Q
apply (erule mp)
  P2 ⊢ P2
apply assumption
done
```

Analysis

- communication classification: one-shot, one-to-many, many-to-one, many-to-many
- control flow analysis
- channel liveness
- algorithm vs constraints
- structural recursion vs fixpoint accumulation
- performance improvements
- safety

Communication Classification

```
structure Serv : SERV =
struct

  datatype serv =
    S of (int * int chan) chan

  fun make () =
  let
    val reqCh = ManyToOne.channel ()
    fun loop state =
    let
      val (v, replCh) = ManyToOne.recv
     reqCh
      val () = OneShot.send (replCh, state
     )
    in
      loop v
    end
    val () = spawn (fn () => loop 0)
  in
    S reqCh
  end
```

```
  fun call (server, v) =
  let
    val S reqCh = server
    val replCh = OneShot.channel ()
    val () = ManyToOne.send (reqCh, (v,
      replCh))
  in
    OneShot.recv replCh
  end

end

val server = Serv.make ()

val () =
  spawn (fn () => Serv.call (server, 35));
  (spawn fn () =>
    Serv.call (server, 12);
    Serv.call (server, 13)
  );
  spawn (fn () => Serv.call (server, 81));
  spawn (fn () => Serv.call (server, 44))
```

Synchronization

- uniprocessor; dispatch scheduling
- multiprocessor; mutex and compare-and-swap
- synchronization state
- sender and receiver thread containers
- message containers

Formal Mechanized Theory

- Isabelle/HOL
- $\sim$ 1421 lines of definitions
- $\sim$ 3052 lines of completed proofs
- syntax-directed rules

Syntax

```
datatype name = Nm string

datatype term =
  Bind name complex term
| Rslt name

and complex =
  Unt
| MkChn
| Atom atom
| Spwn term
| Sync name
| Fst name
| Snd name
| Case name name term name term
| App name name

and atom =
  SendEvt name name
| RecvEvt name
| Pair name name
| Lft name
| Rht name
| Fun name name term
```

Syntax

```
bind u1 = unt
bind r1 = rht u1
bind l1 = lft r1
bind l2 = lft l1

bind mksr = fun _ x2 =>
(
  bind k1 = mkChn
  bind srv = fun srv' x3 =>
  (
    bind e1 = recvEvt k1
    bind p1 = sync e1
    bind v1 = fst p1
    bind k2 = snd p1
    bind e2 = sendEvt k2 x3
    bind z5 = sync e2
    bind z6 = app srv' v1
    rslt z6
  )
  bind z7 = spawn
  (
    bind z8 = app srv r1
    rslt z8
  )
  rslt k1
)
```

```
bind rqst = fun _ x4 =>
(
  bind k3 = fst x4
  bind v2 = snd x4
  bind k4 = mkChn
  bind p2 = pair v2 k4
  bind e3 = sendEvt k3 p2
  bind z9 = sync e3
  bind e4 = recvEvt k4
  bind v3 = sync e4
  rslt v3
)

bind srvr = mksr u1
bind z10 = spawn
(
  bind p3 = pair srvr l1
  bind z11 = app rqst p3
  rslt z11
)
bind p4 = pair srvr l2
bind z12 = app rqst p4
rslt z12
```

Dynamic Semantics

```
datatype dynamic_step =
  DSeq name
| DSpwn name
| DCll name
| DRtn name

type dynamic_path = dynamic_step list

datatype chan =
  Chan dynamic_path name

datatype dynamic_value =
  VUnt
| VChn chan
| VAtm atom (name -> dynamic_value option)

type environment =
  name -> dynamic_value option
```

Dynamic Semantics

```
predicate seqEval: complex -> environment -> dynamic_value -> bool

predicate callEval: complex -> env -> term -> env -> bool

datatype contin = Ctn name tm env

type stack = contin list

datatype state = Stt program env stack

type pool = dynamic_path -> state option

predicate leaf: pool -> dynamic_path -> bool

type corresp = dynamic_path * chan * dynamic_path

type communication = corresp set

predicate dynamicEval: pool -> communication -> pool -> communication -> bool

predicate star: ('a -> 'a -> bool) -> 'a -> 'a -> bool
```

Dynamic Communication

```
predicate isSendPath: pool -> chan -> dynamic_path -> bool

predicate isRecvPath: pool -> chan -> dynamic_path -> bool

predicate forEveryTwo: ('a -> bool) -> ('a -> 'a -> bool) -> bool

predicate ordered: 'a list -> 'a list -> bool

predicate oneToMany: tm -> chan -> bool

predicate manyToOne: tm -> chan -> bool

predicate oneToOne: tm -> chan -> bool

predicate oneShot: tm -> chan -> bool

predicate oneSync: tm -> chan -> bool
```

Dynamic Communication

```
predicate oneToMany: tm -> chan -> bool where
  intro: t0 chan pool comm .
    star dynamicEval [[] -> (Stt t0 [->] [])] {} pool comm,
    forEveryTwo (isSendPath pool chan) ordered
  ⊢ oneToMany t0 chan
```

Static Semantics

```
datatype static_value =
  SUnt
| SChn name
| SAtm atom

type static_value_map =
  name -> static_value set

fun resultName: term -> name

predicate staticEval: static_value_map -> static_value_map -> term -> bool
```

Static Semantics

```
val staticEnv: name -> static_value set =
(
  u1 -> {unt},
  r1 -> {rht u1},
  l1 -> {lft r1},
  l2 -> {lft l1},
  mksr -> {fun _ x2 => ...},
  x2 -> {unt},
  k1 -> {chn k1},
  srv -> {fun srv' x3 => ...},
  srv' -> {fun srv' x3 => ...},
  x3 -> {rht u1, lft r1, lft l1},
  e1 -> {recvEvt k1},
  p1 -> {pair v2 k4},
  v1 -> {lft r1, lft l1},
  k2 -> {chn k4},
  e2 -> {sendEvt k2 x3},
  z5 -> {unt},
  z7 -> {unt},
  u5 -> {unt},
  rqst -> {fun _ x4 => ...},
  x4 -> {pair srvr l1, pair srvr l2},
  k3 -> {chn k1},
  v2 -> {lft r1, lft l1},
  k4 -> {chn k4},
  p2 -> {pair v2 k4},
  e3 -> {sendEvt k3 p2},
  z9 -> {unt},
  e4 -> {recvEvt k4},
  v3 -> {rht u1, lft r1, lft r2},
  srvr -> {chn k1},
  z10 -> {unt},
  p3 -> {pair srvr l1},
  z11 -> {rht u1, lft r2},
  p4 -> {pair srvr l2},
  z12 -> {rht u1, lft l1}
)

val staticComm: name -> static_value set =
(
  k1 -> {pair v2 k4},
  k4 -> {rht u1, lft l1, lft l2}
)
```

Static Communication

```
predicate staticFlowsAccept: static_value_map -> graph -> term -> bool

predicate staticTraceable:
  flow set -> tm_id -> (tm_id -> bool) -> static_path -> bool

predicate staticInclusive: static_path -> static_path -> bool

predicate uncompetitive: static_path -> static_path -> bool

predicate staticOneToMany: term -> name -> bool

predicate staticManyToOne: term -> name -> bool

predicate staticOneToOne: term -> name -> bool

predicate staticOneShot: term -> name -> bool

predicate staticOneSync: term -> name -> bool
```

Static Communication

```
predicate staticOneToMany: term -> name -> bool where
  intro: staticEnv staticComm t graph n_c .
    staticEval staticEnv staticComm t,
    staticFlowsAccept staticEnv graph t,
    forEveryTwo (staticTraceable graph (termId t)
      (staticSendId staticEnv t n_c)) uncompetitive
  ⊢ staticOneToMany t n_c
```

Soundness

- induction on transition system
- generalization to intermediate semantic data structures
- skewing of induction direction
- reversing of inference direction
- preservation

Soundness

```
theorem staticOneToManySound: t0 nc pathc .
  staticOneToMany t0 nc
⊢ oneToMany t0 (Chan pathc nc)

theorem staticManyToOneSound: t0 nc pathc .
  staticManyToOne t0 nc
⊢ manyToOne t0 (Chan pathc nc)

theorem staticOneToOneSound: t0 nc pathc .
  staticOneToOne staticEnv t0 nc
⊢ oneToOne t0 (Chan pathc nc)

theorem staticOneShotSound: t0 nc pathc .
  staticOneShot t0 nc
⊢ oneShot t0 (Chan pathc nc)

theorem staticOneShotSound: t0 nc pathc .
  staticOneSync t0 nc
⊢ oneSync t0 (Chan pathc nc)
```

Soundness

```
lemma staticEvalSound: t₀ pool comm staticEnv staticComm path t env stack n v .
  staticEval staticEnv staticComm t₀,
  star dynamicEval [[] -> (Stt t₀ [->] [])] {} pool comm,
  pool path = Some (Stt t env stack),
  env n = Some v
⊢ abstract v ∈ staticEnv n
```

Higher Precision Analysis

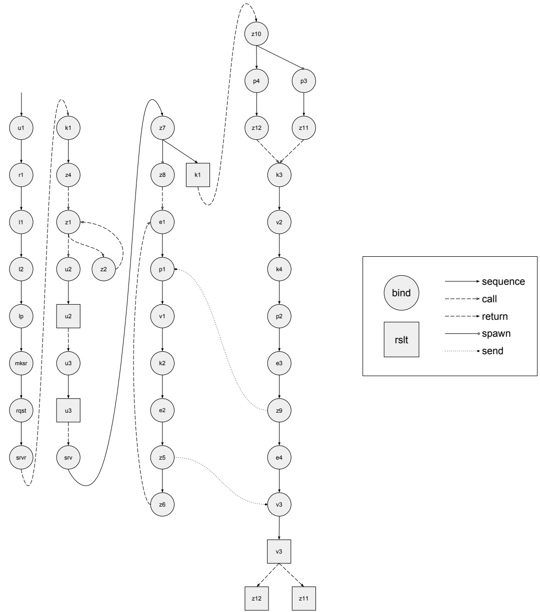
- different channel instances with same name
- channel liveness analysis
- trimmed graph
- paths from channel creation site
- paths along sending transitions

Higher Precision Analysis

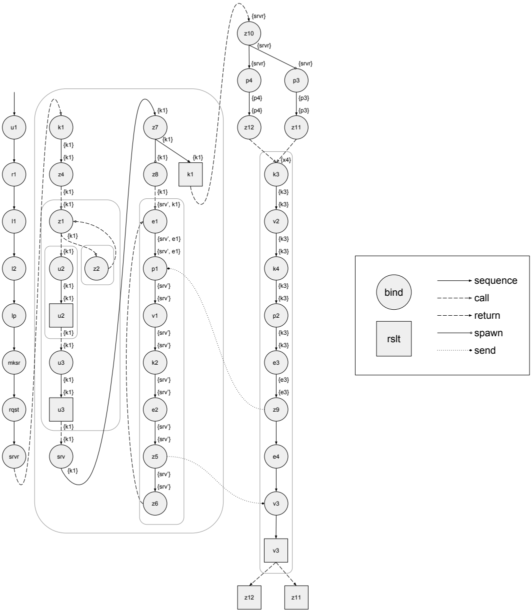
```
predicate staticChanLive:
  static_value_map -> tm_id_map -> tm_id_map -> name -> term -> bool

predicate staticPathLive:
  graph -> tm_id_map -> tm_id_map -> tm_id -> (tm_id -> bool) -> static_path -> bool
```

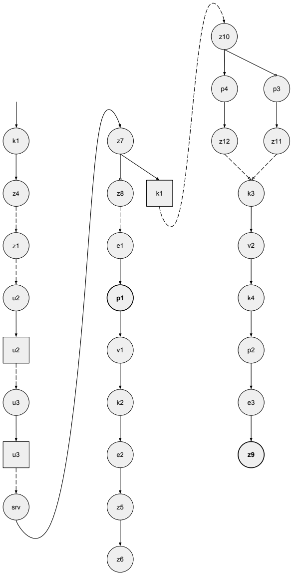
Higher Precision Analysis



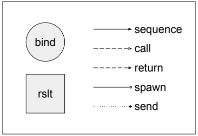
Higher Precision Analysis



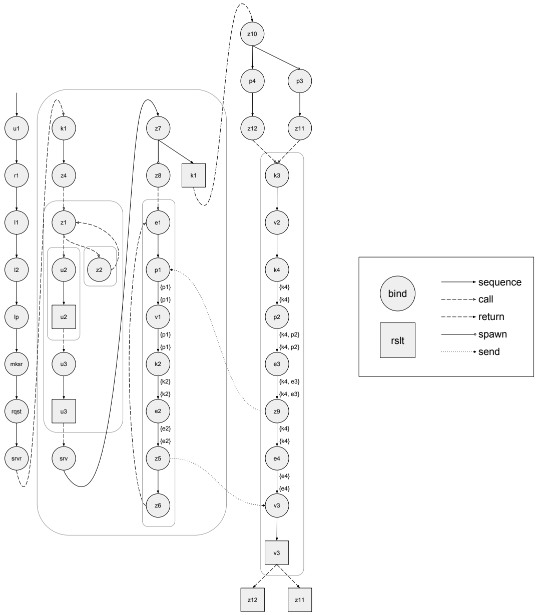
Higher Precision Analysis



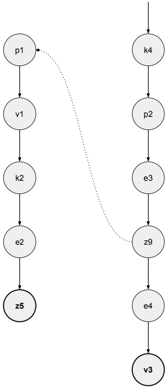

```
chn k1:
  send sites: {bind z9}
  recv sites: {bind p1}
```



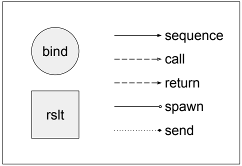
Higher Precision Analysis



Higher Precision Analysis




```
chn k4:
  send sites: {bind z5}
  recv sites: {bind v3}
```



Higher Precision Analysis

```
predicate staticOneToMany: term -> name -> bool where
  intro: staticEnv staticComm t graph entr exit n_c
    staticEval staticEnv staticComm t,
    staticFlowsAccept staticEnv graph t,
    staticChanLive staticEnv entr exit n_c t,
    forEveryTwo (staticPathLive graph entr exit (IdBind n_c)
      (staticSendId staticEnv t n_c)) uncompetitive
  ⊢ staticOneToMany t n_c
```

Summary

- analysis of a simple subset of Concurrent ML
- formal specification: evaluation, communication classifcation, channel liveness
- mechanized soundness proofs: evaluation, communication classifcation
- formal reasoning techniques: generalization, induction skewing, inference direction