

A Mechanized Theory of Communication Analysis in CML

March 4, 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  
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
- ▶ 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

Synchronization

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

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
```

Dynamic Semantics

predicate dynamicEval:

pool -> communication -> pool -> communication -> bool

where

return: pool path n env n_k t_k env_k stack' v comm .

leaf pool path,

pool path = Some (Stt (Rslt n) env ((Ctn n_k t_k env_k) # stack')),

env n = Some v

⊢ dynamicEval

pool comm

(pool(

path @ [DRtn n] ->

(Stt t_k env_k(n_k -> v) stack')

))

comm

* seq: pool path n c t' env stack v .

leaf pool path,

pool path = Some (Stt (Bind n c t') env stack),

seqEval c env v

⊢ dynamicEval

pool comm

(pool(

path @ [DSeq n] -> (Stt t' (env(n -> v)) stack)

))

comm

Dynamic Semantics

```
* call: pool path n c t' env stack tc envc comm .
  leaf pool path,
  pool path = Some (Stt (Bind n c t') env stack),
  callEval c env tc envc
  ⊢ dynamicEval
    pool comm
    (pool(
      path @ [DCll n] -> (Stt tc envc ((Ctn n t' env) # stack))
    )) comm

* makeChan: pool path n t' env stack .
  leaf pool path,
  pool path = Some (Stt (Bind n MkChn t') env stack)
  ⊢ dynamicEval pool comm
    (pool(
      path @ [DSeq n] ->
        (Stt t' (env(n -> (VChn (Chan path n)))) stack)
    )) comm

* spawn: pool path n tc t' env stack comm .
  leaf pool path,
  pool path = Some (Stt (Bind n (Spwn tc) t') env stack)
  ⊢ dynamicEval pool comm
    (pool(
      path @ [DSeq n] -> (Stt t' (env(n -> VUnt)) stack),
      path @ [DSpwn n] -> (Stt tc env [])
    )) comm
```

Dynamic Semantics

```
* sync: pool pathS nS nse tS envS stackS nsc nm
  envse pathR nR nre tR envR stackR nrc envre chan comm .
  leaf pool pathS,
  pool pathS = Some
    (Stt (Bind nS (Sync nse) tS) envS stackS),
  envS nse = Some
    (VAtm (SendEvt nsc nm) envse),
  leaf pool pathR,
  pool pathR = Some
    (Stt (Bind nR (Sync nre) tR) envR stackR),
  envR nre = Some
    (VAtm (RecvEvt nrc) envre),
  envse nsc = Some (VChn chan),
  envre nrc = Some (VChn chan),
  envse nm = Some vm
⊢ dynamicEval
  pool comm
  (pool(
    pathS @ [DSeq nS] -> (Stt tS (envS(nS -> VUnt)) stackS),
    pathR @ [DSeq nR] -> (Stt tR (envR(nR -> vm)) stackR)
  ))
  (comm ∪ {(pathS, chan, pathR)})
```

Dynamic Communication

predicate isSendPath: pool -> chan -> dynamic_path -> bool **where**
intro: pool path n n_e t' env stack n_{sc} n_m env_e chan .
pool path = Some (Stt (Bind n (Sync n_e) t') env stack),
env n_e = Some (VAtm (SendEvt n_{sc} n_m) env_e),
env_e n_{sc} = Some (VChn chan)
 \vdash isSendPath pool chan path

predicate isRecvPath: pool -> chan -> dynamic_path -> bool **where**
intro: pool path n n_e t' env stack n_{rc} env_e chan .
pool path = Some (Stt (Bind n (Sync n_e) t') env stack),
env n_e = Some (VAtm (RecvEvt n_{rc}) env_e),
env_e n_{rc} = Some (VChn chan)
 \vdash isRecvPath pool chan path

predicate forEveryTwo: ('a -> bool) -> ('a -> 'a -> bool) -> bool **where**
intro: p r .
 \forall path1 path2 .
p path1 \wedge p path2 \rightarrow r path1 path2
 \vdash forEveryTwo p r

predicate ordered: 'a list -> 'a list -> bool **where**
first: path1 path2 .
prefix path1 path2
 \vdash ordered path1 path2
* second: path2 path1 .
prefix path2 path1
 \vdash ordered path1 path2

Dynamic Communication

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

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

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

Dynamic Communication

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

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

Static Semantics

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

  result: staticEnv staticComm n .
  ⊢ staticEval staticEnv staticComm (Rslt n)

* unit: staticEnv n staticComm t' .
  SUnt ∈ staticEnv n,
  staticEval staticEnv staticComm t'
  ⊢ staticEval staticEnv staticComm (Bind n Unt t')

* makeChan: n staticEnv staticComm t' .
  (SChn n) ∈ staticEnv n,
  staticEval staticEnv staticComm t'
  ⊢ staticEval staticEnv staticComm (Bind n MkChn t')

* sendEvt: nc nm staticEnv n staticComm t' .
  (SAtm (SendEvt nc nm)) ∈ staticEnv n,
  staticEval staticEnv staticComm t'
  ⊢ staticEval staticEnv staticComm (Bind n (Atom (SendEvt nc nm)) t')

* recvEvt: nc staticEnv n staticComm t' .
  (SAtm (RecvEvt nc)) ∈ staticEnv n,
  staticEval staticEnv staticComm t'
  ⊢ staticEval staticEnv staticComm (Bind n (Atom (RecvEvt nc)) t')
```


Static Semantics

- * pair: $n_1 \ n_2 \ \text{staticEnv} \ n \ \text{staticComm} \ t' \ .$
 $(\text{SATm} (\text{Pair} \ n_1 \ n_2)) \in \text{staticEnv} \ n,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t'$
 $\vdash \text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ (\text{Bind} \ n \ (\text{Atom} \ (\text{Pair} \ n_1 \ n_2)) \ t')$
- * left: $n_a \ \text{staticEnv} \ n \ \text{staticComm} \ t' \ .$
 $(\text{SATm} (\text{Lft} \ n_a)) \in \text{staticEnv} \ n,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t'$
 $\vdash \text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ (\text{Bind} \ n \ (\text{Atom} \ (\text{Lft} \ n_a)) \ t')$
- * right: $n_a \ \text{staticEnv} \ n \ \text{staticComm} \ t' \ .$
 $(\text{SATm} (\text{Rht} \ n_a)) \in \text{staticEnv} \ n,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t'$
 $\vdash \text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ (\text{Bind} \ n \ (\text{Atom} \ (\text{Rht} \ n_a)) \ t')$
- * function: $n_f \ n_t \ t_b \ \text{staticEnv} \ \text{staticComm} \ n \ t' \ .$
 $(\text{SATm} (\text{Fun} \ n_f \ n_t \ t_b)) \in \text{staticEnv} \ n_f,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t_b,$
 $(\text{SATm} (\text{Fun} \ n_f \ n_t \ t_b)) \in \text{staticEnv} \ n,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t'$
 $\vdash \text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ (\text{Bind} \ n \ (\text{Atom} \ (\text{Fun} \ n_f \ n_t \ t_b)) \ t')$
- * spawn: $n_f \ n_t \ t_b \ \text{staticEnv} \ \text{staticComm} \ n \ t' \ .$
 $\text{SUnt} \in \text{staticEnv} \ n,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t_c,$
 $\text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ t'$
 $\vdash \text{staticEval} \ \text{staticEnv} \ \text{staticComm} \ (\text{Bind} \ n \ (\text{Spwn} \ t_c) \ t')$

Static Semantics

- * sync: staticEnv n_e n staticComm t' .
 - $\forall n_{sc} n_m n_c .$
 - $(\text{SAtm } (\text{SendEvt } n_{sc} n_m)) \in \text{staticEnv } n_e$
 - $\rightarrow \text{SChn } n_c \in \text{staticEnv } n_{sc}$
 - $\rightarrow \text{SUnt} \in \text{staticEnv } n \wedge \text{staticEnv } n_m \subseteq \text{staticComm } n_c,$
 - $\forall n_{rc} n_c .$
 - $(\text{SAtm } (\text{RecvEvt } n_{rc})) \in \text{staticEnv } n_e$
 - $\rightarrow \text{SChn } n_c \in \text{staticEnv } n_{rc}$
 - $\rightarrow \text{staticComm } n_c \subseteq \text{staticEnv } n,$
 - staticEval staticEnv staticComm t' $\vdash \text{staticEval staticEnv staticComm } (\text{Bind } n (\text{Sync } n_e) t')$
- * first: staticEnv n_f n staticComm t' .
 - $\forall n_1 n_2 .$
 - $(\text{SAtm } (\text{Pair } n_1 n_2)) \in \text{staticEnv } n_f$
 - $\rightarrow \text{staticEnv } n_1 \subseteq \text{staticEnv } n,$
 - staticEval staticEnv staticComm t' $\vdash \text{staticEval staticEnv staticComm } (\text{Bind } n (\text{Fst } n_f) t')$
- * second: staticEnv n_f n staticComm t' .
 - $\forall n_1 n_2 .$
 - $(\text{SAtm } (\text{Pair } n_1 n_2)) \in \text{staticEnv } n_f$
 - $\rightarrow \text{staticEnv } n_2 \subseteq \text{staticEnv } n,$
 - staticEval staticEnv staticComm t' $\vdash \text{staticEval staticEnv staticComm } (\text{Bind } n (\text{Snd } n_f) t')$

Static Semantics

* distinction: $\text{staticEnv } n_s \ n_l \ t_l \ n \ \text{staticComm } n_r \ t_r \ t' .$

$\forall n_c .$

$(\text{SAtm } (\text{Lft } n_c)) \in \text{staticEnv } n_s$

$\rightarrow \text{staticEnv } n_c \subseteq \text{staticEnv } n_l,$

$\text{staticEnv } (\text{resultName } t_l) \subseteq \text{staticEnv } n,$

$\text{staticEval } \text{staticEnv } \text{staticComm } t_l,$

$\forall n_c .$

$(\text{SAtm } (\text{Rht } n_c)) \in \text{staticEnv } n_s$

$\rightarrow \text{staticEnv } n_c \subseteq \text{staticEnv } n_r,$

$\text{staticEnv } (\text{resultName } t_r) \subseteq \text{staticEnv } n,$

$\text{staticEval } \text{staticEnv } \text{staticComm } t_r,$

$\text{staticEval } \text{staticEnv } \text{staticComm } t'$

$\vdash \text{staticEval } \text{staticEnv } \text{staticComm } (\text{Bind } n \ (\text{Case } n_s \ n_l \ t_l \ n_r \ t_r) \ t')$

* application: $\text{staticEnv } n_f \ n_a \ n \ \text{staticComm } t' .$

$\forall n_f' \ n_t \ t_b .$

$(\text{SAtm } (\text{Fun } n_f' \ n_t \ t_b)) \in \text{staticEnv } n_f$

$\rightarrow \text{staticEnv } n_a \subseteq \text{staticEnv } n_t,$

$\text{staticEnv } (\text{resultName } t_b) \subseteq \text{staticEnv } n,$

$\text{staticEval } \text{staticEnv } \text{staticComm } t'$

$\vdash \text{staticEval } \text{staticEnv } \text{staticComm } (\text{Bind } n \ (\text{App } n_f \ n_a) \ t')$

Static Semantics

```
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
```

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 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  
   $\vdash$  staticOneToMany t  $n_c$ 
```

```
predicate staticManyToOne: term -> name -> bool where  
  intro: staticEnv staticComm t graph  $n_c$  .  
    staticEval staticEnv staticComm t,  
    staticFlowsAccept staticEnv graph t,  
    forEveryTwo (staticTraceable graph (termId t)  
      (staticRecvId staticEnv t  $n_c$ )) uncompetitive  
   $\vdash$  staticManyToOne t  $n_c$ 
```

Static Communication

```
predicate staticOneShot: 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  $n_c$ )) singular  
   $\vdash$  staticOneShot t  $n_c$ 
```

```
predicate staticOneSync: 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  $n_c$ ))  
      singular,  
    forEveryTwo (staticTraceable graph (termId t) (staticRecvId staticEnv t  $n_c$ ))  
      uncompetitive  
   $\vdash$  staticOneSync t  $n_c$ 
```

Soundness

- ▶ induction on transition system
- ▶ generalization for direct derivation
- ▶ equivalent definitions with different structures
- ▶ preservation under transitions

Soundness

theorem staticOneToManySound: $t_0 \ n_C \ \text{path}_C \ .$
staticOneToMany $t_0 \ n_C$

\vdash oneToMany $t_0 \ (\text{Chan } \text{path}_C \ n_C)$

theorem staticManyToOneSound: $t_0 \ n_C \ \text{path}_C \ .$
staticManyToOne $t_0 \ n_C$

\vdash manyToOne $t_0 \ (\text{Chan } \text{path}_C \ n_C)$

theorem staticOneToOneSound: $t_0 \ n_C \ \text{path}_C \ .$
staticOneToOne staticEnv $t_0 \ n_C$

\vdash oneToOne $t_0 \ (\text{Chan } \text{path}_C \ n_C)$

theorem staticOneShotSound: $t_0 \ n_C \ \text{path}_C \ .$
staticOneShot $t_0 \ n_C$

\vdash oneShot $t_0 \ (\text{Chan } \text{path}_C \ n_C)$

theorem staticOneShotSound: $t_0 \ n_C \ \text{path}_C \ .$
staticOneSync $t_0 \ n_C$

\vdash oneSync $t_0 \ (\text{Chan } \text{path}_C \ n_C)$

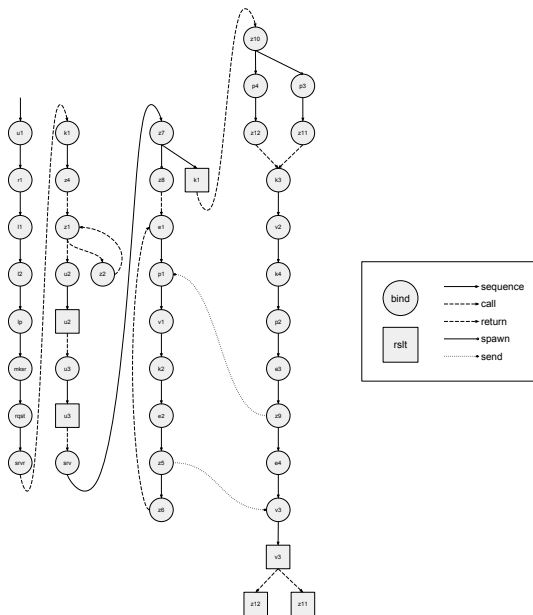
Soundness

```
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
```

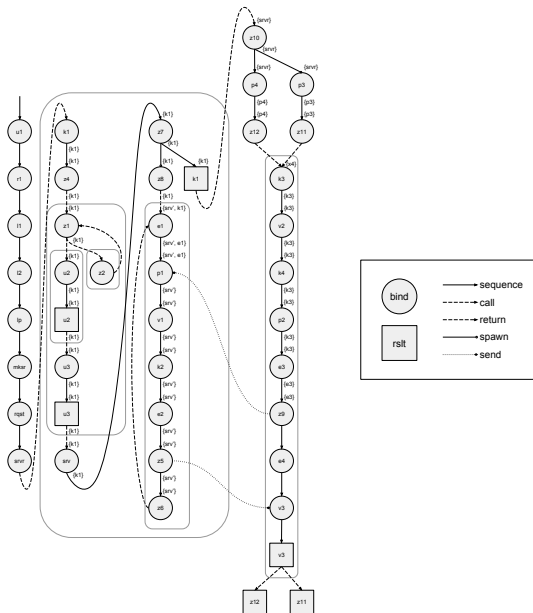
Higher Precision Analysis

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

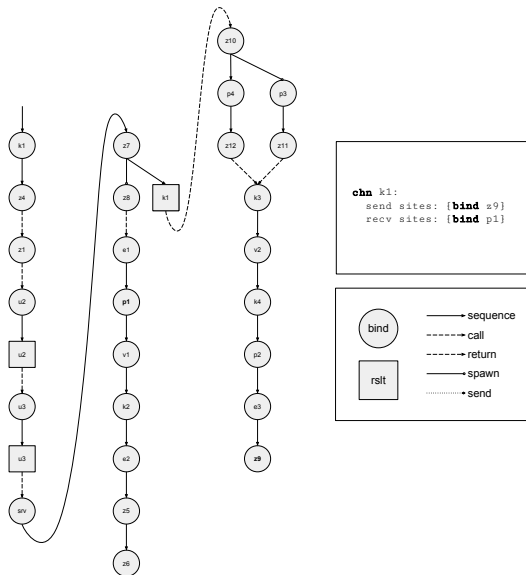
Higher Precision Analysis



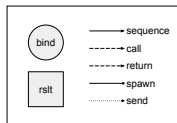
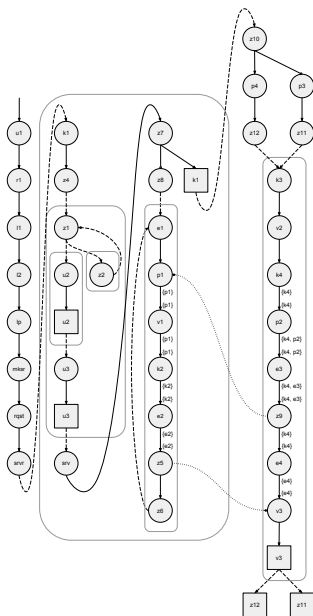
Higher Precision Analysis



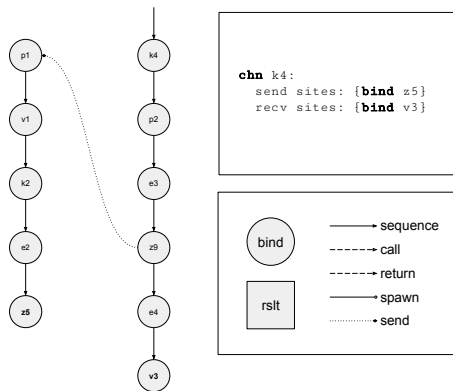
Higher Precision Analysis



Higher Precision Analysis



Higher Precision Analysis



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 (staticTraceable graph entr exit (IdBind  $n_c$ )  
      (staticSendId staticEnv t  $n_c$ )) uncompetitive  
   $\vdash$  staticOneToMany t  $n_c$ 
```

```
predicate staticManyToOne: 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 (staticTraceable graph entr exit (IdBind  $n_c$ )  
      (staticRecvId staticEnv t  $n_c$ )) uncompetitive  
   $\vdash$  staticManyToOne t  $n_c$ 
```

Higher Precision Analysis

```
predicate staticOneToOne: term -> name -> bool where
  intro: staticEnv staticComm t graph entr exit nc
    staticEval staticEnv staticComm t,
    staticFlowsAccept staticEnv graph t,
    staticChanLive staticEnv entr exit nc t,
    forEveryTwo (staticTraceable graph entr exit (IdBind nc)
      (staticSendId staticEnv t nc)) uncompetitive,
    forEveryTwo (staticTraceable graph entr exit (IdBind nc)
      (staticRecvId staticEnv t nc)) uncompetitive
  ⊢ staticOneToOne t nc
```

```
predicate staticOneShot: term -> name -> bool where
  intro: staticEnv staticComm t graph entr exit nc .
    staticEval staticEnv staticComm t,
    staticFlowsAccept staticEnv graph t,
    staticChanLive staticEnv entr exit nc t,
    forEveryTwo (staticTraceable graph entr exit (IdBind nc)
      (staticSendId staticEnv t nc)) singular
  ⊢ staticOneShot t nc
```

Discussion