Formal Theory of Communication in CML

Thomas Logan

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 chan * 'a -> unit event

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

fun recv v = sync (recvEvt v)
```

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
    in
      send (replCh, state); loop v
    end
  in
    spawn (fn () => loop 0); S reqCh
  end

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

```
val server = Serv.make ()

val _ = spawn (fn () =>
  Serv.call (server, 35))

val _ = spawn (fn () =>
  Serv.call (server, 12);
  Serv.call (server, 13))

val _ = spawn (fn () =>
  Serv.call (server, 81))

val _ = spawn (fn () =>
  Serv.call (server, 44))
```

Synchronization Protocol (without event combinators)

single processor sync send_evt:

- two possible transitions:
    - receiver is missing; queued
    - receiver is available; synchronized

multi processor sync send_evt:

- many possible transitions:
    - receiver missing; queued
    - receiver available; claimed
    - receiver claimed; synchronized
    - receiver claimed; sync failed
    - sync failed; receiver missing
    - sync failed; receiver available

multi processor one-to-one sync send_evt:

- same two transitions as general purpose single processor sync

Reppy and Xiao's prototype one-to-one channel on multiprocessor

- 3x - 4x faster than general purpose multi proc implementation
- same speed as general purpose single proc implementation

Manticore's multi processor implementation:

- 2.5x times slower than single processor implementation

Concurrent ML

```
structure Serv :> SERV = struct

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

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

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

Static Communication Topologies

the <u>most</u> number of threads that <u>may</u> compete on channel:

- one-shot:
    - one send may be attempted
- one-sync:
    - one send and one recv may sync overall
- one-to-one: at a given time,
    - one thread may attempt to send (and may sync),
    - one thread may attempt to recv (and may sync)
- fan-out: at a given time,
    - one thread may attempt to send (and may sync),
    - infinite threads may attempt to recv (and may sync)
- fan-in: at a given time,
    - infinite threads may attempt to send (and may sync),
    - one thread may attempt to recv (and may sync)

Correctness Criteria

- clear description of semantics
    - thread pool steps to thread pool
- clear description of static semantics with good precision
    - values may exist at points in a program
- soundness of static semantics
    - if no abstract value exists statically, then no concrete value at runtime
- clear description of communication topologies analysis
    - the number of processes that actually compete on a channel
- clear description of of static communication topologies analysis with good precision
    - on a given channel the most number of processes that may compete on a channel
- soundness of static communication topologies
    - if some number of processes compete on a channel statically, then that number or fewer actually compete at runtime.
- static descriptions can describe precise topologies for typical programs
- precise topologies are computable
- if algorithm produces a topology then topology relation with result is provable
- if the topology relation is provable then the algorithm produces a result that is as precise as that described by the relation

Benefits of Proof Assistant

- automatically proves tedious formulas
- finds lemmas to use
- leads to discovery of errors in definitions
- provides confidence
- lowers burden of extending definitions

Formal Reasoning with Proof Assistants and Theorem Provers

Isabelle

- implemented with and uses Poly/ML
- metalogic used to reason about and define object logics
- the metalogic has syntax:
    - i.e. it's possible to write statements in the metalogic
- Isabelle/HOL
    - one of many implementations of HOL
    - one of many logics defined using Isabelle
    - used in my case to define and reason about a Concurrent ML
    - functions abstract over terms and functions
    - functions are total
    - propositions unified with boolean terms, e.g. True ≡ ((λx::bool. x) = (λx. x)), False ≡ (∀P. P)
    - predicates abstract over terms, functions, propositions, and predicates
    - excluded middle

Coq

- implemented with OCaml
- booleans are not propositions
- propositions are types, proofs are terms
    - i.e. a term is a proof of its type, e.g. 3 is a proof of Nat
- Constructive logic, no excluded middle by default

Formal Reasoning with Proof Assistants and Theorem Provers

Coq:

```
Inductive nat : Type :=
| O : nat
| S : nat → nat.

Inductive ev : nat → Prop :=
| ev_0 : ev 0
| ev_SS : ∀n : nat, ev n → ev (S (S n)).

Fixpoint evn (n:nat) : bool :=
  match n with
    | O ⇒ true
    | S O ⇒ false
    | S (S n') ⇒ evn n'
  end.

Theorem ev_evn :
  ∀(n : nat), ev n → (evn n = true)
```

Isabelle/HOL:

```
datatype nat = O | S nat

inductive ev :: “nat ⇒ bool" where
  ev0:  “ev 0” |
  evSS: “ev n ⟹ ev (S (S n))”

fun evn :: "nat ⇒ bool" where
  “evn O = True” |
  “evn (S O) = False” |
  “evn (S(S n)) = evn n”

lemma “ev m ⟹ evn m”
```

Formal Reasoning with Proof Assistants and Theorem Provers

Coq:

```
Inductive star
  {X:Type}(R: X→X→Prop) : (X→X→Prop) :=
| refl :
  ∀(x : X), star R x x
| step :
  ∀(x y z : X), R x y → star R y z → star R x z.
```

```
Inductive abc (P: Prop): (Prop) :=
  | A : P → abc P.
```

```
Inductive ghi (Prop → Prop) :=
  | G : ghi True.
```

```
Inductive xyz : (Type → Type) :=
  | X : xyz nat.

X : xyz nat
```

Isabelle/HOL:

```
inductive star ::
  “('a ⇒ 'a ⇒ bool) ⇒ 'a ⇒ 'a ⇒ bool”
where
  refl:  “star R x x” |
  step:  “R x y ⟹ star R y z ⟹ star R x z”
```

```
inductive abc :: “bool ⇒ bool” where
  A : “P ⟹ abc P”
```

```
inductive ghi :: “bool ⇒ bool” where
  G : “ghi True”
```

```
datatype ‘a xyz = X ‘a

X (S (S O)) : nat xyz
X ‘’hello’’ : string xyz
```

Syntax

variables   x, y, z, f

expressions   e ::= **let** x **=** b **in** e | **result** x

prims   p ::= **send_evt** x_c x_m | **recv_evt** x_c |
fun f x ⇒ e |
left x | **right** x |
pair x_1 x_2

bindees   b ::= () | p | **chan** () | **spawn** e |
sync x_e | **app** f x |
case x_s **of** x_l ⇒ e_l ¦ x_r ⇒ e_r |
fst x_p | **snd** x_p

Runtime Data Structures and Properties

labels l ::= ˋx | .x | ˧x | ꜔x | ˥x |
˩x

paths π ::= l # π

channels c ::= **ch** π l

values ω ::= () | c | **clos** p ρ

contexts ρ ::= [] | ρ(x ↦ ω)

continuation stack κ ::= [] | ⟨x,e,ρ⟩ # κ

state σ ::= ⟨e,ρ,κ⟩

thread pool T ::= [] | T(π ↦ σ)

Runtime Semantics, part 1

$$\mathtt{T} \rightarrow \mathtt{T'} :$$

$$\frac{\mathbf{leaf}\ \mathtt{T}\ \pi \qquad \mathtt{T}\ \pi = \langle \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathtt{()}\ \mathbf{in}}{\mathtt{T} \rightarrow \mathtt{T}(\pi;;\backprime\mathtt{x} \mapsto \langle \mathtt{e},\rho(\mathtt{x} \mapsto \mathtt{()}),\kappa\rangle)}\ \mathtt{unit}$$

$$\frac{\mathbf{leaf}\ \mathtt{T}\ \pi \qquad \mathtt{T}\ \pi = \langle \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathtt{p}\ \mathbf{in}\ \mathtt{e},\rho,\kappa\rangle}{\mathtt{T} \rightarrow \mathtt{T}(\pi;;\backprime\mathtt{x} \mapsto \langle \mathtt{e},\rho(\mathtt{x} \mapsto \mathtt{clos\ p}\ \rho),\kappa\rangle)}\ \mathtt{prim}$$

$$\frac{\mathbf{leaf}\ \mathtt{T}\ \pi \qquad \mathtt{T}\ \pi = \langle \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{chan}\ \mathtt{()}\ \mathbf{in}\ \mathtt{e},\rho,\kappa\rangle)}{\mathtt{T} \rightarrow \mathtt{T}(\pi;;\backprime\mathtt{x} \mapsto \langle \mathtt{e},\rho(\mathtt{x} \mapsto \mathbf{ch}\ \pi\ \mathtt{x}),\kappa\rangle)}\ \mathtt{chan}$$

$$\frac{\mathbf{leaf}\ \mathtt{T}\ \pi \qquad \mathtt{T}\ \pi = \langle \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{spawn}\ \mathtt{e_c}\ \mathbf{in}\ \mathtt{e},\rho,\kappa\rangle}{\begin{array}{l}\mathtt{T} \rightarrow \mathtt{T}(\\ \quad \pi;;\backprime\mathtt{x} \mapsto \langle \mathtt{e},\rho(\mathtt{x} \mapsto \mathbf{()}),\kappa\rangle,\ \pi;;.\mathtt{x} \mapsto \langle \mathtt{e_c},\rho,[\,]\rangle \\ )\end{array}}\ \mathtt{spawn}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ \mathtt{T}\ \pi_s \qquad \mathtt{T}\ \pi_s = \langle \mathbf{let}\ \mathtt{x_s}\ \mathbf{=}\ \mathbf{sync}\ \mathtt{x_{se}}\ \mathbf{in}\ \mathtt{e_s},\rho_s,\kappa_s\rangle \\ \rho_s\ \mathtt{x_{se}} = \mathbf{clos}\ (\mathbf{send\_evt}\ \mathtt{x_{sc}}\ \mathtt{x_m})\ \rho_{se} \\ \mathbf{leaf}\ \mathtt{T}\ \pi_r \qquad \mathtt{T}\ \pi_r = \langle \mathbf{let}\ \mathtt{x_r}\ \mathbf{=}\ \mathbf{sync}\ \mathtt{x_{re}}\ \mathbf{in}\ \mathtt{e_r},\rho_r,\kappa_r\rangle \\ \rho_r\ \mathtt{x_{re}} = \mathbf{clos}\ (\mathbf{recv\_evt}\ \mathtt{x_{rc}})\ \rho_{re} \\ \rho_s\ \mathtt{x_{sc}} = \mathtt{c} \qquad \rho_r\ \mathtt{x_{rc}} = \mathtt{c} \qquad \rho_{se}\ \mathtt{x_{sc}} = \omega_m\end{array}}{\begin{array}{l}\mathtt{T} \rightarrow \mathtt{T}(\\ \quad \pi_s;;\backprime\mathtt{x_s} \mapsto \langle \mathtt{e_s},\rho_s(\mathtt{x_s} \mapsto \mathtt{()}),\kappa\rangle, \\ \quad \pi_r;;\backprime\mathtt{x_r} \mapsto \langle \mathtt{e_r},\rho_r(\mathtt{x_r} \mapsto \omega_m),\kappa\rangle \\ )\end{array}}\ \mathtt{sync}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ \mathtt{P}\ \pi \qquad \mathtt{P}\ \pi = \langle \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{app}\ \mathtt{f}\ \mathtt{x_a}\ \mathbf{in}\ \mathtt{e},\rho,\kappa\rangle \\ \rho\ \mathtt{f} = \mathbf{clos}\ (\mathbf{fun}\ \mathtt{f'}\ \mathtt{x_p} \Rightarrow \mathtt{e_b})\ \rho' \qquad \rho\ \mathtt{x_a} = \omega\end{array}}{\begin{array}{l}\mathtt{T} \rightarrow \mathtt{T}(\pi;;\upharpoonleft\mathtt{x} \mapsto \langle \mathtt{e_b},\rho'(\\ \quad \mathtt{f'} \mapsto \mathbf{clos}\ (\mathbf{fun}\ \mathtt{f'}\ \mathtt{x_p} \Rightarrow \mathtt{e_b})\ \rho', \\ \quad \mathtt{x_p} \mapsto \omega \\ ),\ \langle \mathtt{x},\mathtt{e},\rho\rangle\ \#\ \kappa\rangle)\end{array}}\ \mathtt{app}$$

Runtime Semantics, part 2

$T \to T'$:

$$\frac{\begin{array}{l}\mathbf{leaf}\ T\ \pi \\ T\ \pi = \langle \mathbf{let}\ x\ \mathbf{=}\ \mathbf{case}\ x_s\ \mathbf{of}\ x_l \Rightarrow e_l \ \ldots\ \mathbf{in}\ e,\rho,\kappa\rangle \\ \rho\ x_s = \mathbf{clos}\ (\mathbf{left}\ x')\ \rho' \qquad \rho'\ x' = \omega\end{array}}{T \to T(\pi;;\dashv x \mapsto \langle e_l,\rho(x_l \mapsto \omega),\langle x,e,\rho\rangle\#\kappa\rangle)}\ \text{case\_l}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ T\ \pi \\ T\ \pi = \langle \mathbf{let}\ x\ \mathbf{=}\ \mathbf{case}\ x_s\ \ldots\ ¦\ x_r \Rightarrow e_r\ \mathbf{in}\ e,\rho,\kappa\rangle \\ \rho\ x_s = \mathbf{clos}\ (\mathbf{right}\ x')\ \rho' \qquad \rho'\ x' = \omega\end{array}}{T \to T(\pi;;\vdash x \mapsto \langle e_r,\rho(x_r \mapsto \omega),\langle x,e,\rho\rangle\#\kappa\rangle)}\ \text{case\_r}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ T\ \pi \qquad T\ \pi = \langle \mathbf{let}\ x\ \mathbf{=}\ \mathbf{fst}\ x_p\ \mathbf{in}\ e,\rho,\kappa\rangle \\ \rho\ x_p = \mathbf{clos}\ (\mathbf{pair}\ x_1\ x_2)\ \rho' \qquad \rho'\ x_1 = \omega\end{array}}{T \to T(\pi;;\grave{}\,x \mapsto \langle e,\rho(x \mapsto \omega),\kappa\rangle)}\ \text{fst}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ T\ \pi \qquad T\ \pi = \langle \mathbf{let}\ x\ \mathbf{=}\ \mathbf{snd}\ x_p\ \mathbf{in}\ e,\rho,\kappa\rangle \\ \rho\ x_p = \mathbf{clos}\ (\mathbf{pair}\ x_1\ x_2)\ \rho' \qquad \rho'\ x_2 = \omega\end{array}}{T \to T(\pi;;\grave{}\,x \mapsto \langle e,\rho(x \mapsto \omega),\kappa\rangle)}\ \text{snd}$$

$$\frac{\begin{array}{l}\mathbf{leaf}\ T\ \pi \qquad T\ \pi = \langle \mathbf{result}\ x,\rho,\langle x',e',\rho'\rangle\#\kappa\rangle \\ \rho\ x = \omega\end{array}}{T \to T(\pi;;\downharpoonleft x \mapsto \langle e',\rho'(x' \mapsto \omega),\kappa\rangle)}\ \text{result}$$

Runtime Communication Topology Analysis, part 1

$\textbf{is\_send\_path}\ T\ c\ \pi$:

$$\frac{\begin{array}{l} T\ \pi = \langle \textbf{let}\ x\ \textbf{=}\ \textbf{sync}\ x_e\ \textbf{in}\ e_{n,}\ \rho,\ \kappa\rangle \\ \rho\ x_e = \textbf{clos}\ (\textbf{send\_evt}\ x_{sc}\ x_m)\ \rho_e \\ \rho_e\ x_{sc} = c \qquad T\ (\pi;;\texttt{`}x) = \langle e_n, \rho(x \mapsto ()), \end{array}}{\textbf{is\_send\_path}\ T\ c\ (\pi;;\texttt{`}x)}$$

$\textbf{is\_recv\_path}\ T\ c\ \pi$:

$$\frac{\begin{array}{l} T\ \pi = \langle \textbf{let}\ x\ \textbf{=}\ \textbf{sync}\ x_e\ \textbf{in}\ e_n,\ \rho,\ \kappa\rangle \\ \rho\ x_e = \textbf{clos}\ (\textbf{recv\_evt}\ x_{sc})\ \rho_e \\ \rho_e\ x_{sc} = c \qquad T\ (\pi;;\texttt{`}x) = \langle e_n, \rho(x \mapsto \omega), \kappa\rangle \end{array}}{\textbf{is\_recv\_path}\ T\ c\ (\pi;;\texttt{`}x)}$$

Runtime Communication Topology Analysis, part 2

```
one_sync T c :

all (is_send_path T c) equal
all (is_recv_path T c) equal
──────────────────────────────
one_sync T c


one_to_one T c :

all (is_send_path T c) ordered
all (is_recv_path T c) ordered
────────────────────────────────
one_to_one T c
```

```
fan_out T c :

all (is_send_path T c) ordered
────────────────────────────────
fan_out T c


fan_in T c :

all (is_recv_path T c) ordered
────────────────────────────────
fan_in T c
```

Static Data Structures

abstract values	$\hat{\omega}$ ::= () | p | **ch** x
environments	V ::= [_ ↦ {}] | V(x ↦ {ω,…})
result variables	⌊e⌋ = x : ⌊**result** x⌋ = x ⌊**let** x **=** b **in** e⌋ = ⌊e⌋
value abstraction	‖ω‖ = $\hat{\omega}$: ‖()‖ = () ‖clos p ρ‖ = p ‖ch π x‖ = ch x
context abstraction	‖ρ‖ = V : ‖ρ‖ = λω.‖ρ ω‖

Static Semantics, part 1

$V\ C \vDash e :$

$$\frac{\{()\} \subseteq V\ x \qquad V\ C \vDash e}{V\ C \vDash \textbf{let}\ x = ()\ \textbf{in}\ e}\ \text{unit}$$

$$\frac{\{\textbf{send\_evt}\ x_c\ x_m\} \subseteq V\ x \qquad V\ C \vDash e}{V\ C \vDash \textbf{let}\ x = \textbf{send\_evt}\ x_c\ x_m\ \textbf{in}\ e}\ \text{send\_evt}$$

$$\frac{\{\textbf{recv\_evt}\ x_c\ x_m\} \subseteq V\ x \qquad V\ C \vDash e}{V\ C \vDash \textbf{let}\ x = \textbf{recv\_evt}\ x_c\ x_m\ \textbf{in}\ e}\ \text{recv\_evt}$$

$$\frac{\begin{array}{ll}\{\textbf{fun}\ f\ x_p \Rightarrow e_b\} \subseteq V\ f & V\ C \vDash e_b \\ \{\textbf{fun}\ f\ x_p \Rightarrow e_b\} \subseteq V\ x & V\ C \vDash e\end{array}}{V\ C \vDash \textbf{let}\ x = \textbf{fun}\ f\ x_p \Rightarrow e_b\ \textbf{in}\ e}\ \text{fun}$$

$$\frac{\{\text{chan}\ x\} \subseteq V\ x \qquad V\ C \vDash e}{V\ C \vDash \textbf{let}\ x = \text{chan}\ ()\ \textbf{in}\ e}\ \text{chan}$$

$$\frac{\{()\} \subseteq V\ x \qquad V\ C \vDash e_c \qquad V\ C \vDash e}{V\ C \vDash \textbf{let}\ x = \textbf{spawn}\ e_c\ \textbf{in}\ e}\ \text{spawn}$$

$$\frac{\begin{array}{l}(\forall\ x_{sc}\ x_m\ x_c\ . \\ \ \textbf{send\_evt}\ x_{sc} \in V\ x_e \longrightarrow \\ \ \textbf{ch}\ x_c \in V\ x_{sc} \longrightarrow \\ \ \{()\} \subseteq V\ x\ \wedge\ V\ x_m \subseteq C\ x_c) \\ \\ (\forall\ x_{rc}\ x_c\ . \\ \ \textbf{recv\_evt}\ x_{rc} \in V\ x_e \longrightarrow \\ \ \textbf{ch}\ x_c \in V\ x_{rc} \longrightarrow \\ \ C\ x_c \subseteq V\ x) \\ \\ V\ C \vDash e\end{array}}{V\ C \vDash \textbf{let}\ x = \textbf{sync}\ x_e\ \textbf{in}\ e}\ \text{sync}$$

Static Semantics, part 2

$\mathrm{V\ C} \vDash \mathrm{e}$:

$$
\dfrac{\begin{array}{l}(\forall\ f'\ x_p\ e_b\ .\\ \ \ \mathbf{fun}\ f'\ x_p \Rightarrow e_b \in V\ f \longrightarrow\\ \ \ V\ x_a \subseteq V\ x_p \wedge V\ (\lfloor e_b \rfloor) \subseteq V\ x)\\ \\ V\ C \vDash e\end{array}}{V\ C \vDash \mathbf{let}\ x = \mathbf{app}\ f\ x_a\ \mathbf{in}\ e}\ \text{app}
$$

$$
\dfrac{\begin{array}{l}(\forall\ x_l'\ .\\ \ \ \mathbf{left}\ x_l' \in V\ x_s \longrightarrow\\ \ \ V\ x_l' \subseteq V\ x_l \wedge V\ (\lfloor e_l \rfloor) \subseteq V\ x \wedge (V,\ C) \vDash e_l)\\ \\ (\forall\ x_r'\ .\\ \ \ \mathbf{right}\ x_r' \in V\ x_s \longrightarrow\\ \ \ V\ x_r' \subseteq V\ x_r \wedge V\ (\lfloor e_r \rfloor) \subseteq V\ x \wedge (V,\ C) \vDash e_r)\\ \\ V\ C \vDash e\end{array}}{V\ C \vDash \mathbf{let}\ x = \mathbf{case}\ x_s\ \mathbf{of}\ x_l \Rightarrow e_l \mid x_r \Rightarrow e_r\ \mathbf{in}\ e}\ \text{case}
$$

$$
\dfrac{\begin{array}{l}\forall\ x_1\ x_2\ .\ \mathbf{pair}\ x_1\ x_2 \in V\ x_p \longrightarrow V\ x_1 \subseteq V\ x\\ V\ C \vDash e\end{array}}{V\ C \vDash \mathbf{let}\ x = \mathbf{fst}\ x_p\ \mathbf{in}\ e}\ \text{fst}
$$

$$
\dfrac{\begin{array}{l}\forall\ x_1\ x_2\ .\ \mathbf{pair}\ x_1\ x_2 \in V\ x_p \longrightarrow V\ x_2 \subseteq V\ x\\ V\ C \vDash e\end{array}}{V\ C \vDash \mathbf{let}\ x = \mathbf{snd}\ x_p\ \mathbf{in}\ e}\ \text{fst}
$$

$$
\dfrac{}{V\ C \vDash \mathbf{result}\ x}\ \text{result}
$$

Static Semantics, part 3

$V\ C \vDash \omega$:

$$\frac{}{V\ C \vDash ()}\ \texttt{unit}$$

$$\frac{\textbf{fun}\ f\ x \Rightarrow e \subseteq V\ f \qquad V\ C \vDash e \qquad V\ C \vDash \rho}{V\ C \vDash \textbf{clos}\ (\textbf{fun}\ f\ x \Rightarrow e)\ \rho}\ \texttt{fun}$$

$$\frac{}{V\ C \vDash \textbf{ch}\ x}\ \texttt{chan}$$

$$\frac{V\ C \vDash \rho}{V\ C \vDash \textbf{clos}\ (\textbf{send\_evt}\ x_c\ x_m)\ \rho}\ \texttt{send\_evt}$$

$V\ C \vDash \rho$:

$$\frac{\forall\ x\ \omega\ .\ \rho\ x = \omega \longrightarrow \{\|\omega\|\} \subseteq V\ x \wedge V\ C \vDash \omega}{V\ C \vDash \rho}$$

$V\ C \vDash W \Rrightarrow \kappa$:

$$\frac{W \subseteq V\ x \qquad V\ C \vDash e \qquad V\ C \vDash \rho \qquad V\ C \vDash V\ \lfloor e \rfloor \Rrightarrow \kappa}{V\ C \vDash W \Rrightarrow \langle x, e, \rho\rangle \# \kappa}$$

$V\ C \vDash \sigma$:

$$\frac{V\ C \vDash e \qquad V\ C \vDash \rho \qquad V\ C \vDash V\ \lfloor e \rfloor \Rrightarrow \kappa}{V\ C \vDash \langle e;\ \rho;\ \kappa\rangle}$$

$V\ C \vDash T$:

$$\frac{\forall\ \pi\ \sigma\ .\ T\ \pi = \sigma \longrightarrow V\ C \vDash \sigma}{V\ C \vDash T}$$

Static Semantics

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathtt{e}\ \mathbf{:}}{\mathtt{V}\ \mathtt{e}_0 \vdash [\texttt{`}\mathtt{x}_0] \mapsto \mathtt{e}_0}\ \texttt{start}$$

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathtt{b}\ \mathbf{in}\ \mathtt{e}}{\mathtt{V}\ \mathtt{e}_0 \vdash \pi\mathbf{;;}\texttt{`}\mathtt{x} \mapsto \mathtt{e}}\ \texttt{seq}$$

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{spawn}\ \mathtt{e}_c\ \mathbf{in}\ \mathtt{e}}{\mathtt{V}\ \mathtt{e}_0 \vdash \pi\mathbf{;;}.\mathtt{x} \mapsto \mathtt{e}_c}\ \texttt{spawn}$$

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{app}\ \mathtt{f}\ \mathtt{x}\ \mathbf{in}\ \mathtt{e} \qquad \mathbf{fun}\ \mathtt{f}'\ \mathtt{x}_p \Rightarrow \mathtt{e}_b \in \mathtt{V}\ \mathtt{f}}{\mathtt{V}\ \mathtt{e}_0 \vdash \pi\mathbf{;;}{\upharpoonleft}\mathtt{x} \mapsto \mathtt{e}_b}\ \texttt{app}$$

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{case}\ \mathtt{x}_s\ \mathbf{of}\ \mathtt{x}_l \Rightarrow \mathtt{e}_l\ \ldots\ \mathbf{in}\ \mathtt{e}}{\mathtt{V}\ \mathtt{e}_0 \vdash \pi\mathbf{;;}{\dashv}\mathtt{x} \mapsto \mathtt{e}_l}\ \texttt{case\_l}$$

$$\frac{\mathtt{V}\ \mathtt{e}_0 \vdash \pi \mapsto \mathbf{let}\ \mathtt{x}\ \mathbf{=}\ \mathbf{case}\ \mathtt{x}_s\ \ldots\ ¦\ \mathtt{x}_r \Rightarrow \mathtt{e}_r\ \mathbf{in}\ \mathtt{e}}{\mathtt{V}\ \mathtt{e}_0 \vdash \pi\mathbf{;;}{\vdash}\mathtt{x} \mapsto \mathtt{e}_r}\ \texttt{case\_r}$$

Static Semantics

```
V e₀ ⊢ π @ ↿x # π' ↦ result y     bal π'
V e₀ ⊢ π ↦ let x = b in e
──────────────────────────────────────────── result_app
V e₀ ⊢ π @ ↿x # (π';;⇃x) ↦ e
```

```
V e₀ ⊢ π @ ↿x # π' ↦ result y     bal π'
V e₀ ⊢ π ↦ let x = b in e
──────────────────────────────────────────── res_cas_l
V e₀ ⊢ π @ ┤x # (π';;⇃x) ↦ e
```

```
V e₀ ⊢ π @ ↿x # π' ↦ result y     bal π'
V e₀ ⊢ π ↦ let x = b in e
──────────────────────────────────────────── res_case_r
V e₀ ⊢ π @ ├x # (π';;⇃x) ↦ e
```

Static Semantics

$$\mathtt{V}\ e_0 \vdash \pi \mapsto \kappa\ \mathbf{:}$$

$$\frac{\mathbf{bal}\ \pi_2}{\mathtt{V}\ e_0 \vdash \pi_1\ @\ .x\ \#\ \pi_2 \mapsto [\,]}\ \mathtt{empty}$$

$$\frac{\begin{array}{l}\mathtt{V}\ e_0 \vdash \pi_1 \mapsto \mathbf{let}\ x\ \mathbf{=}\ \mathbf{app}\ f\ x_a\ \mathbf{in}\ e \\ \mathbf{bal}\ \pi_2 \qquad \mathtt{V}\ e_0 \vdash \pi_1 \mapsto \kappa\end{array}}{\mathtt{V}\ e_0 \vdash \pi_1\ @\ \upharpoonleft x\ \#\ \pi_2 \mapsto \langle x,e,\rho\rangle\ \#\ \kappa}\ \mathtt{app}$$

$$\frac{\begin{array}{l}\mathtt{V}\ e_0 \vdash \pi_1 \mapsto \mathbf{let}\ x\ \mathbf{=}\ \mathbf{case}\ x_s\ \mathbf{of}\ x_l \Rightarrow e_l\ \ldots\ \mathbf{in}\ e \\ \mathbf{bal}\ \pi_2 \qquad\qquad \mathtt{V}\ e_0 \vdash \pi_1 \mapsto \kappa\end{array}}{\mathtt{V}\ e_0 \vdash \pi_1\ @\ \dashv x\ \#\ \pi_2 \mapsto \langle x,e,\rho\rangle\ \#\ \kappa}\ \mathtt{case\_l}$$

$$\frac{\begin{array}{l}\mathtt{V}\ e_0 \vdash \pi_1 \mapsto \mathbf{let}\ x\ \mathbf{=}\ \mathbf{case}\ x_s\ \ldots\ \mid\ x_r \Rightarrow e_r\ \mathbf{in}\ e \\ \mathbf{bal}\ \pi_2 \qquad\qquad \mathtt{V}\ e_0 \vdash \pi_1 \mapsto \kappa\end{array}}{\mathtt{V}\ e_0 \vdash \pi_1\ @\ \vdash x\ \#\ \pi_2 \mapsto \langle x,e,\rho\rangle\ \#\ \kappa}\ \mathtt{case\_r}$$

Static Communication Topology Analysis, part 1

$\textbf{is\_static\_send\_path}\ \mathtt{V}\ \mathtt{C}\ \mathtt{e}\ \mathtt{x}\ \pi$ **:**

$$\frac{\begin{array}{l} \mathtt{V}\ e_0 \vdash \pi \mapsto \textbf{let}\ x\ \textbf{=}\ \textbf{sync}\ x_e\ \textbf{in}\ e_n \\ \{\textbf{send\_evt}\ x_{sc}\ x_m\} \subseteq \mathtt{V}\ x_e \qquad \{\textbf{ch}\ x_c\} \subseteq \mathtt{V}\ x_{sc} \\ \{()\} \subseteq \mathtt{V}\ x_{sc} \qquad \mathtt{V}\ x_m \subseteq \mathtt{C}\ x_c \end{array}}{\textbf{is\_static\_send\_path}\ \mathtt{V}\ \mathtt{C}\ e_0\ x_c\ (\pi;;\grave{}x)}$$

$\textbf{is\_static\_recv\_path}\ \mathtt{V}\ \mathtt{C}\ \mathtt{e}\ \pi$ **:**

$$\frac{\begin{array}{l} \mathtt{V}\ e_0 \vdash \pi \mapsto \textbf{let}\ x\ \textbf{=}\ \textbf{sync}\ x_e\ \textbf{in}\ e_n \\ \{\textbf{recv\_evt}\ x_{rc}\} \subseteq \mathtt{V}\ x_e \qquad \{\textbf{ch}\ x_c\} \subseteq \mathtt{V}\ x_{sc} \\ \{\hat{\omega}\} \subseteq \mathtt{V}\ x_{rc} \qquad \{\hat{\omega}\} \subseteq \mathtt{C}\ x_c \end{array}}{\textbf{is\_static\_recv\_path}\ \mathtt{V}\ \mathtt{C}\ e_0\ x_c\ (\pi;;\grave{}x)}$$

$\textbf{exclusive}\ \pi_1\ \pi_2$ **:**

$$\frac{\pi_1 = \pi_2}{\textbf{exclusive}\ \pi_1\ \pi_2}$$

$\textbf{noncompet}\ \pi_1\ \pi_2$ **:**

$$\frac{\textbf{ordered}\ \pi_1\ \pi_2 \lor \pi_1 \cong \pi_2}{\textbf{noncompet}\ \pi_1\ \pi_2}$$

Static Communication Topology Analysis, part 2

$$\mathbf{static\_one\_sync}\ V\ C\ e\ x :$$

$$\frac{\mathbf{all}\ (\mathbf{is\_static\_send\_path}\ V\ C\ e\ x_c)\ \mathbf{exclusive} \qquad \mathbf{all}\ (\mathbf{is\_static\_recv\_path}\ V\ C\ e\ x_c)\ \mathbf{exclusive}}{\mathbf{static\_one\_sync}\ V\ C\ e\ x_c}$$

$$\mathbf{static\_one\_to\_one}\ V\ C\ e\ x :$$

$$\frac{\mathbf{all}\ (\mathbf{is\_static\_send\_path}\ V\ C\ e\ x_c)\ \mathbf{noncompet} \qquad \mathbf{all}\ (\mathbf{is\_static\_recv\_path}\ V\ C\ e\ x_c)\ \mathbf{noncompet}}{\mathbf{static\_one\_to\_one}\ V\ C\ e\ x_c}$$

$$\mathbf{fan\_out}\ V\ C\ e\ x :$$

$$\frac{\mathbf{all}\ (\mathbf{is\_static\_send\_path}\ V\ C\ e\ x_c)\ \mathbf{noncompet}}{\mathbf{static\_fan\_out}\ V\ C\ e\ x_c}$$

$$\mathbf{static\_fan\_in}\ V\ C\ e\ x :$$

$$\frac{\mathbf{all}\ (\mathbf{is\_static\_recv\_path}\ V\ C\ e\ x_c)\ \mathbf{noncompet}}{\mathbf{static\_fan\_in}\ V\ C\ e\ x_c}$$

Semantics Theorems, part 1

```
theorem static_semantics_preserved :

V C ⊨ T       T → T'
————————————————————————
V C ⊨ T′

proof by inversion on → and on ⊨ and subsequent ⊨


theorem static_semantics_preserved_star :

T →* T'
————————————————————————
V C ⊨ T ⟶ V C ⊨ T′

proof:
  1.  case T →* T' ≡ T →* T :
        a.  T = T′
        b.  V C ⊨ T ⟶ V C ⊨ T
        c.  V C ⊨ T ⟶ V C ⊨ T′
  2.  case T →* T' ≡ T → Tm ∧ Tm →* T′ :
        a.  V C ⊨ Tm ⟶ V C ⊨ T′ (IH)
        b.  V C ⊨ T ⟶ V C ⊨ Tm by preservation under T → Tm
        c.  V C ⊨ T ⟶ V C ⊨ T′ by transitive implication
qed by induction on →*
```

Semantics Theorems, part 2

```
theorem runtime_evaluation_more_precise :

T π = ⟨e,ρ,κ⟩    V C ⊨ T
────────────────────────
‖ρ‖ ⊑ V

proof by inversion on ⊨ and subsequent ⊨


theorem not_evaluation/sound :

[[.x₀] ↦ ⟨e,[],[]⟩] →* T′    T′ π = ⟨e',ρ',κ'⟩
V C ⊨ e
──────────────────────────────────────────────
‖ρ′‖ ⊑ V

proof:
  1.  V C ⊨ [[.x₀] ↦ ⟨e;Map.empty;[]⟩] by construction
  2.  V C ⊨ T′ by preservation of ⊨ under →*
  3.  ‖ρ′‖ ⊑ V by precision under ⊨
qed
```

Semantics Theorems, part 3

theorem **exp_traceable_preserved :**

$$\frac{(\forall \pi\ e\ \rho\ \kappa.\ T\ \pi = \langle e,\rho,\kappa\rangle \longrightarrow V\ e_0 \vdash \pi \mapsto e \wedge V\ e_0 \vdash \pi \mapsto \kappa) \quad V\ C \vDash T \quad T \rightarrow T'}{T'\ \pi' = \langle e',\rho',\kappa'\rangle \longrightarrow V\ e_0 \vdash \pi' \mapsto e'}$$

proof by inversion of → , construction of ⊢ π ↦ e,
and additional preservation properties

theorem **stack_traceable_preserved :**

$$\frac{(\forall \pi\ e\ \rho\ \kappa.\ T\ \pi = \langle e,\rho,\kappa\rangle \longrightarrow V\ e_0 \vdash \pi \mapsto e \wedge V\ e_0 \vdash \pi \mapsto \kappa) \quad V\ C \vDash T \quad T \rightarrow T'}{T'\ \pi' = \langle e',\rho',\kappa'\rangle \longrightarrow V\ e_0 \vdash \pi' \mapsto \kappa'}$$

proof by inversion of → , construction of ⊢ π ↦ κ,
and additional preservation properties

Semantics Theorems, part 4

theorem **not_traceable/sound_strong :**

$T_0 \rightarrow^* T'$

$T_0 = [[.x_0] \mapsto \langle e_0,[],[]\rangle] \longrightarrow V\ C \vDash T_0 \longrightarrow$
$(\forall\ \pi'\ e'\ \rho'\ \kappa'\ .$
$\quad T'\ \pi' = \langle e',\rho',\kappa'\rangle \longrightarrow$
$\quad V\ e_0 \vdash \pi' \mapsto e' \wedge V\ e_0 \vdash \pi' \mapsto \kappa')$

proof:

1. $T_0\ {}^*\!\!\rightarrow T$ by star equivalence
2. case $T_0\ {}^*\!\!\rightarrow T' \equiv T_0\ {}^*\!\!\rightarrow T_0$:
   a. $T' = T_0$
   b. assume and intro
   c. $\pi = [.x_0] \wedge e = [] \wedge \kappa = []$
   d. $V\ e_0 \vdash [.x_0] \mapsto e_0 \wedge V\ e_0 \vdash [.x_0] \mapsto []$ by start and empty rules
   e. $V\ e_0 \vdash \pi \mapsto e \wedge V\ e_0 \vdash \pi \mapsto \kappa$
3. case $T_0\ {}^*\!\!\rightarrow T' \equiv T_0\ {}^*\!\!\rightarrow T \wedge T \rightarrow T'$:
   a. $T\ \pi = \langle e,\rho,\kappa\rangle \longrightarrow V\ e_0 \vdash \pi \mapsto e \wedge V\ e_0 \vdash \pi \mapsto \kappa$ by IH
   b. $T_0 \rightarrow^* T$ by star equivalence
   c. $V\ C \vDash T$ by preservation of $\vDash$ under $\rightarrow^*$
   d. $V\ e_0 \vdash \pi' \mapsto e' \wedge V\ e_0 \vdash \pi' \mapsto \kappa'$ by preservation under $\vDash$ and $\rightarrow$

qed

Semantics Theorems, part 5

theorem **not_traceable/sound :**

$$\frac{[[.x_0] \mapsto \langle e_0,[],[]\rangle] \to^* T \qquad T\ \pi = \langle e,\rho,\kappa\rangle}{V\ C \vDash e_0 \longrightarrow V\ e_0 \vdash \pi \mapsto e}$$

proof:

1. $V\ C \vDash [[.x_0] \mapsto \langle e,[],[]\rangle]$ by construction
2. $V\ e_0 \vdash \pi \mapsto e$ by not_traceable/sound_strong

qed

Communication Topology Theorems

theorem **not_recv_path/sound :**

$$\frac{\texttt{[[.}x_0\texttt{]} \mapsto \langle e,[\,],[\,]\rangle\texttt{]} \rightarrow^{*} T' \qquad \textbf{is\_recv\_path}\ T\ (\textbf{ch}\ \pi_c\ x_c)\ \pi}{V\ C \vDash e \longrightarrow \textbf{is\_static\_recv\_path}\ V\ C\ e\ x_c\ \pi}$$

proof using not_evaluation/sound and not_traceable/sound

Communication Topology Theorems

theorem **recv_paths_ordered/sound :**

$$
\begin{array}{l}
\mathtt{V\ C} \vDash \mathtt{e} \\
(\forall\ \pi_1\ \pi_2\ . \\
\quad \textbf{is\_static\_recv\_path}\ \mathtt{V\ C\ e}\ x_c\ \pi_1 \longrightarrow \\
\quad \textbf{is\_static\_recv\_path}\ \mathtt{V\ C\ e}\ x_c\ \pi_2 \longrightarrow \\
\quad \textbf{noncompetitive}\ \pi_1\ \pi_2 \\
) \\
[[.x_0] \mapsto \langle \mathtt{e},[\,],[\,] \rangle] \rightarrow^* \mathtt{T} \\
\dfrac{\textbf{is\_recv\_path}\ \mathtt{T}\ (\textbf{ch}\ \pi_c\ x_c)\ \pi_1 \qquad \textbf{is\_recv\_path}\ \mathtt{T}\ (\textbf{ch}\ \pi_c\ x_c)\ \pi_2}{\textbf{ordered}\ \pi_1\ \pi_2}
\end{array}
$$

```
proof using not_recv_path/sound
   and runtime_paths_ordered_or_nonexclusive and
```

Communication Topology Theorems

theorem **fan_in/sound :**

$$\frac{V\ C \vDash e \qquad \textbf{static\_fan\_in}\ V\ C\ e\ x_c}{[[.x_0] \mapsto \langle e,[\,],[\,]\rangle] \rightarrow^{*} T' \longrightarrow \textbf{fan\_in}\ T'\ (\textbf{ch}\ \pi\ x_c)}$$

proof by recv_paths_ordered/sound