Static analysis and all that

Martin Steffen

IfI UiO

Spring 2016

Plan

- approx. 15 lectures, details see web-page
- flexible time-schedule, depending on progress/interest
- covering parts/following the structure of textbook [Nielson et al., 1999], concentrating on
  - overview
  - data-flow
  - control-flow
  - type- and effect systems
- on request, new parts possible
- helpful prior knowledge: having at least heard of
  - typed lambda calculi (especially for CFA)
  - simple type systems
  - operational semantics
  - lattice theory, fixpoints, induction

  but things needed will be covered . . .

1. Types and effects
    - Control flow analysis
    - Side effect analysis
    - Exception analysis
    - Regions
    - Behavior

Introduction

- now: working with a
  - typed language
  - functional language Fun
- cf. the corresponding section in the intro (annotated types)
- here: control-flow analysis (perhaps more). Remember also the constraint based analysis/CFA in the intro

Syntax

$$
\begin{array}{rcl}
e & ::= & c \mid x \mid \mathtt{fn}_\pi x \Rightarrow e \mid \mathtt{fun}_\pi f\ x \Rightarrow e \mid e\ e \qquad \text{terms} \\
& \mid & \mathtt{if}\ e\ \mathtt{then}\ e\ \mathtt{else}\ e \mid \mathtt{let}\ x = e\ \mathtt{in}\ e \mid e\ \mathtt{op}\ e
\end{array}
$$

Table: Abstract syntax

π	$\in$	**Pnt**	program points
e	$\in$	**Expr**	expressions
c	$\in$	**Const**	constants
$\mathtt{op}$	$\in$	**Op**	operators
f, x	$\in$	**Var**	variables

Example (application)

$$(\mathtt{fn}_X\, x \Rightarrow x)\ (\mathtt{fn}_Y\, y \Rightarrow y)$$

Example

$$\begin{array}{lll} \mathtt{let}\; g & = (\mathtt{fun}_F\, f\ x \Rightarrow f(\mathtt{fn}_Y\, y \Rightarrow y)) \\ \mathtt{in} & g\ (\mathtt{fn}_Z\, x \Rightarrow x) \end{array}$$

Types

- Curry-style typing

$$\begin{array}{lll} \tau & \in & \textbf{Type} \quad \text{types} \\ \Gamma & \in & \textbf{TEnv} \quad \text{type environment} \end{array}$$

Types

$$\tau ::= \mathsf{int} \mid \mathsf{bool} \mid \tau \to \tau$$

- base types:
  - bool and int
  - standard constants and operators assumed (true, 5, $+$, $\leq$, $\ldots$)
  - each constant has a base[1] type τ_c
- type environments (finite mapping)[2]

$$\Gamma ::= [] \mid \Gamma, x{:}\tau$$

[1] restricting assumption (no higher-order constants, only operations).

[2] We can also write $\Gamma[x \mapsto \tau]$, and use $dom(\Gamma)$ and $\Gamma(x)$. Γ acts like a stack.

Judgments and derivation system

type judgments

$$\Gamma \vdash_{\mathrm{UL}} e : \tau$$

- derivation system:
  - Curry-style formulation
  - ⇒ non-deterministic
  - nonetheless: monomorphic let[3]
- type reconstruction/type inference

[3] This remark is relevant only for the ones who know, that a famous contribution was polymorphic let.

$$\Gamma \vdash c : \tau_c \quad \textsc{Con} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textsc{Var}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \texttt{fn}_\pi x \Rightarrow e : \tau_1 \to \tau_2}\ \textsc{Fn} \qquad \frac{\Gamma, x{:}\tau_1, f{:}\tau_1 \to \tau_2 \vdash e : \tau_2}{\Gamma \vdash \texttt{fun}_\pi x \Rightarrow e : \tau_1 \to \tau_2}\ \textsc{Fun}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

$$\frac{\Gamma \vdash e_0 : \text{bool} \qquad \Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}\ \textsc{If}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x{:}\tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Let}$$

$$\frac{\Gamma \vdash e_1 : \tau^1_{\mathrm{op}} \qquad \Gamma \vdash e_2 : \tau^2_{\mathrm{op}}}{\Gamma \vdash e_1\ \mathrm{op}\ e_2 : \tau_{\mathrm{op}}}\ \textsc{Op}$$

Control flow analysis

- remember introduction: CFA touched 2 times
  - constraint analysis
  - as effect-system ("call-tracking")
- goal CFA (general): "which functions may be applied in an application"
- more precisely:

CFA

to which function abstractions may an expression *evaluate to*

- augment/annotate the type system with effects
- note: data "=" control here

Annotations

annotations: set of function names

$$\begin{array}{lcll} \varphi & \in & \textbf{Ann} & \text{annotations} \\ \hat{\tau} & \in & \widehat{\textbf{Type}} & \text{annotated types} \\ \hat{\Gamma} & \in & \widehat{\textbf{TEnv}} & \text{ann. type environments} \end{array}$$

$$\varphi ::= \{\pi\} \mid \varphi \cup \varphi \mid \emptyset$$

$$\hat{\tau} ::= \mathsf{int} \mid \mathsf{bool} \mid \hat{\tau} \xrightarrow{\varphi} \hat{\tau}$$

$$\hat{\Gamma} ::= [] \mid \hat{\Gamma}, x{:}\hat{\tau}$$

Erasure to underlying TS: $\lfloor\hat{\tau}\rfloor$, $\lfloor\hat{\Gamma}\rfloor$

$$\hat{\Gamma} \vdash c : \tau_c \quad \textsc{Con} \qquad\qquad \frac{\hat{\Gamma}(x) = \hat{\tau}}{\hat{\Gamma} \vdash x : \hat{\tau}}\ \textsc{Var}$$

$$\frac{\hat{\Gamma}, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2}{\hat{\Gamma} \vdash\ \texttt{fn}_\pi x \Rightarrow e : \hat{\tau}_1 \stackrel{\{\pi\}\cup\varphi}{\to} \hat{\tau}_2}\ \textsc{Fn} \qquad \frac{\Gamma, x{:}\hat{\tau}_1, f{:}\hat{\tau}_1 \stackrel{\{\pi\}\cup\varphi}{\to} \hat{\tau}_2 \vdash e : \hat{\tau}_2}{\Gamma \vdash\ \texttt{fun}_\pi x \Rightarrow e : \hat{\tau}_1 \stackrel{\{\pi\}\cup\varphi}{\to} \hat{\tau}_2}\ \textsc{Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 \stackrel{\varphi}{\to} \hat{\tau}_2 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_1}{\hat{\Gamma} \vdash e_1\ e_2 : \hat{\tau}_2}\ \textsc{App}$$

$$\frac{\hat{\Gamma} \vdash e_0 : \mathsf{bool} \qquad \hat{\Gamma} \vdash e_1 : \hat{\tau} \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}}{\Gamma \vdash\ \texttt{if}\, e_0\, \texttt{then}\, e_1\, \texttt{else}\, e_2 : \hat{\tau}}\ \textsc{If}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 \qquad \Gamma, x{:}\hat{\tau}_1 \vdash e_2 : \hat{\tau}_2}{\hat{\Gamma} \vdash\ \texttt{let}\, x = e_1\, \texttt{in}\, e_2 : \hat{\tau}_2}\ \textsc{Let}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \tau_{\mathrm{op}}^1 \qquad \hat{\Gamma} \vdash e_2 : \tau_{\mathrm{op}}^2}{\hat{\Gamma} \vdash e_1\, \mathrm{op}\, e_2 : \tau_{\mathrm{op}}}\ \textsc{Op}$$

Example

$$\dfrac{\dfrac{x{:}\hat{\tau}_Y \vdash x{:}\hat{\tau}_Y}{\vdash (\underset{X}{\texttt{fn}}\, x \Rightarrow x) : \hat{\tau}_Y \overset{\{X\}}{\to} \hat{\tau}_Y} \qquad \dfrac{y{:}\mathrm{int} \vdash y : \mathrm{int}}{\vdash (\underset{Y}{\texttt{fn}}\, y \Rightarrow y) : \hat{\tau}_Y}}{\vdash (\underset{X}{\texttt{fn}}\, x \Rightarrow x)\ (\underset{Y}{\texttt{fn}}\, y \Rightarrow y) : \hat{\tau}_Y}$$

with

$$\hat{\tau}_Y = \mathrm{int} \overset{\{Y\}}{\to} \mathrm{int}$$

Example: loop

page 287

Equivalence of annotations

- the annotations φ are considered as sets
- one could axiomatise this (UCAI)
- i.e., one could import equality on sets into equality on types:

$$\frac{\hat{\tau}_1 = \hat{\tau}_1' \qquad \hat{\tau}_2 = \hat{\tau}_2' \qquad \varphi = \varphi'}{\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 = \hat{\tau}_1' \xrightarrow{\varphi'} \hat{\tau}_2'}$$

- types (and $\hat{\Gamma}$'s) are considered only modulo this equality
- seems like a minor/innocent/obvious point, but: it gives serious technic headaches when we go for *type reconstruction/inference!*

Properties

- desired relationship between the original type system and the annotated one:

$\Rightarrow$ The annotation does not “disturb” the original one

- conservative extension
- note:
  - type systems reject programs
  - flow analysis and similar: typically don’t reject, just analyse

Properties

- desired relationship between the original type system and the annotated one:

⇒ The annotation does not "disturb" the original one

- conservative extension
- note:
  - type systems reject programs
  - flow analysis and similar: typically don't reject, just analyse

Fact

- *if* $\hat{\Gamma} \vdash_{CFA}: e : \hat{\tau}$ *then* $\lfloor\hat{\Gamma}\rfloor \vdash_{UL} e : \lfloor\hat{\tau}\rfloor$
- *if* $\Gamma \vdash_{UL} e : \tau$, *then* $\hat{\Gamma} \vdash_{CFA} e : \hat{\tau}$ *for some* $\hat{\Gamma}$ *and* $\hat{\tau}$ *s.t.* $\tau = \lfloor\hat{\tau}\rfloor$ *and* $\Gamma = \lfloor\hat{\Gamma}\rfloor$.

- note: the role of the "liberal" rules for abstraction

Natural semantics

- now: natural semantics = big-step semantics
- unlike before (e.g.: while-language): small-step semantics

transitions

$$\vdash e \longrightarrow v$$

"expression e reduces/evaluates to *value* v"

- assume: e is closed
- values $v \in \mathbf{Val}$

$$v ::= c \mid \ \mathtt{fn}_\pi x \Rightarrow e \quad \text{prov. that } fv(\mathtt{fn}_\pi x \Rightarrow e) = \emptyset$$

- note:
  - no bind (and close) construct[4]
  - substitution can be literal

[4] This remark is only relevant if you have read other parts, or if you know what bind/close is.

$$\vdash c \longrightarrow c \quad \textsc{Con} \qquad \vdash \ \mathtt{fn}_\pi x \Rightarrow e \longrightarrow \ \mathtt{fn}_\pi x \Rightarrow e \quad \textsc{Fn}$$

$$\vdash \ \mathtt{fun}_\pi f\ x \Rightarrow e \longrightarrow \ \mathtt{fn}_\pi x \Rightarrow (e[\mathtt{fun}_\pi f\ x \Rightarrow e/f]) \quad \textsc{Fun}$$

$$\frac{\vdash e_1 \longrightarrow \ \mathtt{fn}_\pi x \Rightarrow e_1' \qquad \vdash e_2 \longrightarrow v_2 \qquad \vdash e_1'[x/v_2] \longrightarrow v}{\vdash e_1\ e_2 \longrightarrow v}\ \textsc{App}$$

$$\frac{\vdash e_0 \longrightarrow \mathsf{true} \qquad \vdash e_1 \longrightarrow v_1}{\vdash \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \longrightarrow v_1}\ \textsc{If}_1$$

$$\frac{\vdash e_1 \longrightarrow v_1 \qquad e_2[v_1/x] \longrightarrow v_2}{\vdash \mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \longrightarrow v_2}\ \textsc{Let}$$

$$\frac{\vdash e_1 \longrightarrow v_1 \qquad \vdash e_2 \longrightarrow v_2 \qquad v_1\ \mathbf{op}\ v_2 = v}{\vdash e_1\ \mathtt{op}\ e_2 \longrightarrow v}\ \textsc{Op}$$

Example

$$h \triangleq \; \texttt{fun}_F f\ x \Rightarrow f\ (\texttt{fn}_Y y \Rightarrow y)$$

$$\dfrac{\vdash h \longrightarrow \texttt{fn}_F x \Rightarrow h\ id_y \qquad \dfrac{\ldots \qquad \dfrac{\vdash h \longrightarrow \texttt{fn}_F x \Rightarrow h\ id_y \ldots \quad \dfrac{\vdots}{\vdash h\ id_y \longrightarrow}}{\vdash h\ id_y \longrightarrow}}{\vdash (\texttt{fn}_F x \Rightarrow h\ id_y)\ id_z \longrightarrow}\ \textsc{App}}{\vdash \texttt{let}\ g = h\ \texttt{in}\ g\ id_z \longrightarrow}$$

typical for “big-step”: Non-termination of program corresponds to diverging derivation tree

Small-step semantics

- another common form of operational semantics
- expecially common in imperative setting (and basically needed in concurrent settings)
- here
  - more complex formulation as usual
  - a simpler formulation semantics possible
    - non-deterministic reduction (but confluent)[5]
    - substitution-based
  - for technical reasons:
    - abstractions carry labels
    - semantics: intended to "preserve" labels
    - ⇒ closure-based
    - Remember also: small

[5] Purely functional/declarative language.

Small-step semantics

$$\frac{v = \rho(x)}{\rho \vdash x \to v}\ \textsc{Var}$$

$$\frac{\rho_0 = \rho\downarrow_{fv(\texttt{fn}\, x \Rightarrow e)}}{\rho \vdash \texttt{fn}\, x \Rightarrow e \to \texttt{close}(\texttt{fn}\, x \Rightarrow e)\,\texttt{in}\,\rho_0}\ \textsc{Fn}$$

$$\frac{\rho_0 = \rho\downarrow_{fv(\texttt{fun}\, f\ x \Rightarrow e)}}{\rho \vdash \texttt{fn}\, f\ x \Rightarrow e \to \texttt{close}(\texttt{fun}\, f\ x \Rightarrow e)\,\texttt{in}\,\rho_0}\ \textsc{Fun}$$

$$\frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash ie_1\ ie_2 \to ie_1'\ ie_2}\ \textsc{App}_1 \qquad \frac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash v\ ie_2 \to v\ ie_2'}\ \textsc{App}_2$$

Small-step semantics

$$\frac{}{\rho \vdash (\texttt{close}(\texttt{fn}\, x \Rightarrow e_1)\,\texttt{in}\,\rho_1)\ v_2 \rightarrow \text{bind}\ \rho_1[x \mapsto v_2]\ \texttt{in}\ e_1}\ \textsc{App}_{fn}$$

$$\frac{\rho_2 = \rho_1[f \mapsto \texttt{close}(\texttt{fun}\, f\ x \Rightarrow e_1)\,\texttt{in}\,\rho_1]}{\rho \vdash (\texttt{close}(\texttt{fn}\, x \Rightarrow e_1)\,\texttt{in}\,\rho_2)\ v_2 \rightarrow \text{bind}\ \rho_2[x \mapsto v_2]\ \texttt{in}\ e_1}\ \textsc{App}_{fun}$$

$$\frac{\rho_1 \vdash ie_1 \rightarrow ie_1'}{\rho \vdash \text{bind}\ \rho_1\ \texttt{in}\ ie_1 \rightarrow \text{bind}\ \rho_1\ \texttt{in}\ ie_1'}\ \text{BIND}_1$$

$$\frac{}{\rho \vdash \text{bind}\ \rho_1\ \texttt{in}\ v_1 \rightarrow v_1}\ \text{BIND}_2$$

Small-step semantics

$$\frac{\rho \vdash ie_0 \longrightarrow ie_0'}{\rho \vdash \texttt{if}\ ie_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \longrightarrow \texttt{if}\ ie_0'\ \texttt{then}\ e_1\ \texttt{else}\ e_2}\ \textsc{If}_0$$

$$\frac{}{\rho \vdash \texttt{if}\ \textsf{true}\ \texttt{then}\ e_2\ \texttt{else}\ e_2 \rightarrow e_1}\ \textsc{If}_1$$

$$\frac{\rho \vdash ie_1 \longrightarrow ie_1'}{\rho \vdash \texttt{let}\ x = ie_1\ \texttt{in}\ e_2 \rightarrow \texttt{let}\ x = ie_1'\ \texttt{in}\ e_2}\ \textsc{Let}_1$$

$$\frac{\rho_0 = \rho\downarrow_{fv(e_2)}}{\rho \vdash \texttt{let}\ x = v\ \texttt{in}\ e_2 \rightarrow \textsf{bind}\ \rho_0[x \mapsto v]\ \texttt{in}\ e_2}\ \textsc{Let}_2$$

Small-step semantics

$$\frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash ie_1 \ \mathrm{op}\ ie_2 \rightarrow ie_1' \ \mathrm{op}\ ie_2}\ \textsc{Op}_1 \qquad \frac{\rho \vdash ie_2 \rightarrow ie_2'}{\rho \vdash v_1 \ \mathrm{op}\ ie_2 \rightarrow v_1 \ \mathrm{op}\ ie_2'}\ \textsc{Op}_2$$

$$\frac{v = v_1 \ \mathbf{op}\ v_2}{\rho \vdash v_1 \ \mathrm{op}\ v_2 \rightarrow v}\ \textsc{Op}_3$$

Semantic correctness

- as always: the analysis as overapproximation
- formulated here (as basically always) as “subject reduction”
- Assume: typing for op is given

Theorem

If $[] \vdash_{\mathrm{CFA}} e : \hat{\tau}$ *and* $\vdash e \longrightarrow v$, *then* $[] \vdash_{\mathrm{CFA}} v : \hat{\tau}$

key lemma: preservation under substitution

Lemma (Substitution)

Assume $[] \vdash_{\mathrm{CFA}} e_0 : \hat{\tau}_0$ *and* $\hat{\Gamma}[x \mapsto \hat{\tau}_0] \vdash_{\mathrm{CFA}} e : \hat{\tau}$. *Then*
$\hat{\Gamma} \vdash_{\mathrm{CFA}} e[x \mapsto e_0] : \hat{\tau}$

Semantics correctness & subject reduction

- “subject reduction”: standard name for key to correctess (aka type safety) in static type systems (here type and effects)

Goal (“Milner’s dictum”)

A well-typed program cannot go wrong.

- goal a bit more technically: no “erroneous” state is reachable, starting from a/the initial state
- erroneous state: a state where a run-time type error manifests itself
  - wrong argument
  - data stored in variable not declared/dimensioned to hold that kind of data
  - “method not supported”
  - ...

Type safety

- as said: subject reduction key
- type safety: statement about all reachable states

⇒ inductive proof

Type safety: 3 easy pieces

- Induction: all reachable “states” are well-typed

  Base case: The initial state is well-typeed

  Induction: Well-typedness is preserved under doing a step (= subject reduction)

- a well-type state is not erroneous at that point

- base case trivial/by assumption: only well-typed programs are run
- since well-typing is preserved: no run-time type checks needed (efficiency)
- with effects (CFA no effects yet): subject reduction = simulation

Complete lattice of annotated types

- to assure existence of solutions

$$(\mathbf{Ann}, \sqsubseteq) \qquad (\simeq (2^{\mathbf{Pnt}}, \subseteq))$$

- write $\hat{\mathbf{Type}}(\tau)$: set of $\hat{\tau}$'s s.t. $\lfloor\hat{\tau}\rfloor = \tau$

$$\frac{}{\hat{\tau} \sqsubseteq \hat{\tau}} \qquad \frac{\hat{\tau}_1 \sqsubseteq \hat{\tau}_1' \qquad \hat{\tau}_2 \sqsubseteq \hat{\tau}_2' \qquad \varphi \sqsubseteq \varphi'}{\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 \sqsubseteq \hat{\tau}_1' \xrightarrow{\varphi'} \hat{\tau}_2'}$$

$\Rightarrow$

$(\hat{\mathbf{Type}}(\tau), \sqsubseteq)$ is complete lattice

Moore family and existence of solutions

- remember: laxness (non-determinism) of the analysis
- typical for specification of analyses (as opposed to algorithms
- "required" condition for "meaningful" annotation/analyses

$$\mathrm{JUDG}_{\mathrm{CFA}}[\Gamma \vdash_{\mathrm{UL}} e : \tau]$$

set of $\hat{\Gamma} \vdash_{\mathrm{CFA}} e : \hat{\tau}$ s.t. $\lfloor \_ \rfloor$ maps ("strips") it to $\Gamma \vdash_{\mathrm{UL}} e : \tau$

Lemma

$\mathrm{JUDG}_{\mathrm{CFA}}[\Gamma \vdash_{\mathrm{UL}} e : \tau]$ *is a Moore family whenever* $\Gamma \vdash_{\mathrm{UL}} e : \tau$

- sounds complex but
  - simple sanity condition
  - existence of solutions
  - "flow analyses" analyze, but don't reject programs (unlike types)
  - unique minimal solution

Inference algorithms

- take care of terminology
- so far: no algorithm! (price of laxness)
- foresight needed
- guessing wrong ⇒ backtracking (and we seriously don't want that)

⇒ required: mechanism to make
  - tentative guesses
  - refine guesses

- we start first: with the underlying system

Augmented types

$$\begin{array}{lll} \tau & \in & \textbf{AType} \quad \text{augmented types} \\ \alpha & \in & \textbf{TVar} \quad \text{type variables} \end{array}$$

$$\begin{array}{lll} \tau & ::= & \mathsf{int} \mid \mathsf{bool} \mid \tau \rightarrow \tau \mid \alpha \\ \alpha & ::= & '\mathtt{a} \mid '\mathtt{b} \mid \ldots \end{array}$$

- fancy name for: "we have added type variables"

Substitution (in general)

mapping from variables to “terms”

- “syntactic mapping”[6]
- here:
  - “terms” are (augmented) types
  - variables: type variables

$$\theta : \textbf{TVar} \rightarrow_{\mathit{fin}} \textbf{AType}$$

considered as finite functions: we write $dom(\theta)$.
Sometimes: considered as total functions, setting $\theta(\alpha) = \alpha$ when $\alpha \notin dom(\theta)$.

- ground substitution: mapping to ordinary types
- substitutions: lifted to types in the standard manner
- composition of substitutions: $\theta_1 \circ \theta_2$ (or just $\theta_1\theta_2$)

[6] A state maps variables to (semantic) values, instead.

Algorithm: basic idea

- instead of guessing type now $\Rightarrow$ postpone the decision

$\Rightarrow$ use of type variables

replace:

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \ \texttt{fn}_\pi x \Rightarrow e : \tau_1 \rightarrow \tau_2}\ \textsc{Fn}$$

by

Algorithm: basic idea

- instead of guessing type now $\Rightarrow$ postpone the decision

$\Rightarrow$ use of type variables

$$\frac{\Gamma, x{:}\alpha \vdash e : \tau_2}{\Gamma \vdash \ \mathtt{fn}_\pi x \Rightarrow e : \alpha \rightarrow \tau_2} \ \textsc{Fn}$$

Algorithm: basic idea

- instead of guessing type now ⇒ postpone the decision

⇒ use of type variables

$$\frac{\Gamma, x{:}\alpha \vdash e : \tau_2}{\Gamma \vdash \mathtt{fn}_\pi x \Rightarrow e : \alpha \to \tau_2} \text{ FN}$$

- $x{:}\alpha$ when α is fresh (otherwise unused) means: type of x is completely arbitrary.
- syntax-directed now?
- τ_1: meta-variable for concrete types
- α: (still meta variable for) type variables

Algorithm: basic idea

- instead of guessing type now $\Rightarrow$ postpone the decision

$\Rightarrow$ use of type variables

α's completely arbitrary?
Consider body

$$e = x\ g$$

for $\texttt{fn}_\pi x \Rightarrow e$
$\Rightarrow$

Restriction on α

- a function type: $\alpha = \beta \rightarrow \gamma$
- fit together with type of g $\Rightarrow$ condition or constraint on β

Algorithm: basic idea

- instead of guessing type now ⇒ postpone the decision

⇒ use of type variables

- judments "give back" not just the type, but also "restrictions" on type variables.
- represented as substitutions[7]
- ⇒

$$\Gamma \vdash e : (\tau, \theta)$$

Under the assumptions Γ (which might "assign" to (program) variables: type variables), program e possesses type τ (again potentially containing type variables) *and* imposes the restrictions "embodied" by θ on the type variables.

[7] One could also collect the constraints/restrictions as a set of *equations* and solve them at the very end.

Algorithm: basic idea

- instead of guessing type now ⇒ postpone the decision

⇒ use of type variables

$$\frac{\Gamma, x{:}\alpha \vdash e_0 : (\tau_0, \theta_0)}{\Gamma \vdash \mathtt{fn}_\pi x \Rightarrow e :} \textsc{Fn}$$

Algorithm: basic idea

- instead of guessing type now ⇒ postpone the decision

⇒ use of type variables

$$\frac{\Gamma, x{:}\alpha \vdash e_0 : (\tau_0, \theta_0)}{\Gamma \vdash \mathtt{fn}_\pi x \Rightarrow e : ((\theta_0 \alpha) \to \tau_0,\ \theta_0)} \text{ FN}$$

$$\frac{}{\Gamma \vdash c : (\tau_c, id)}\ \textsc{T-Const} \qquad \frac{}{\Gamma \vdash x : (\Gamma(x), id)}\ \textsc{T-Var}$$

$$\frac{\alpha \text{ fresh} \qquad \Gamma, x{:}\alpha \vdash e_0 : (\tau_0, \theta_0)}{\Gamma \vdash \ \texttt{fn}_\pi x \Rightarrow e_0 : (\theta_0\alpha \rightarrow \tau_0, \theta_0)}\ \textsc{T-Fn}$$

$$\frac{\alpha, \alpha_0 \text{ fresh} \quad \Gamma, f{:}\alpha \rightarrow \alpha_0, x{:}\alpha \vdash e_0 : (\tau_0, \theta_0) \quad \theta_1 = \mathcal{U}(\tau_0, \theta_0\alpha_0)}{\Gamma \vdash \texttt{fun}_\pi f\ x \Rightarrow e_0 : (\theta_1\theta_0\alpha \rightarrow \theta_1(\tau_0), \theta_1 \circ \theta_0)}\ \textsc{T-Fun}$$

$$\frac{\Gamma \vdash e_1 : (\tau_1, \theta_1) \quad \theta_1\Gamma \vdash e_2 : (\tau_2, \theta_2) \quad \alpha \text{ fresh} \quad \theta_3 = \mathcal{U}(\theta_2\tau_1, \tau_2 \rightarrow \alpha)}{\Gamma \vdash e_1\ e_2 : (\theta_3\alpha, \theta_3\theta_2\theta_1)}\ \textsc{T-App}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_0 : (\tau_0, \theta_0) \qquad \theta_0\Gamma \vdash e_1 : (\tau_1, \theta_1) \qquad \theta_1\theta_0\Gamma \vdash e_2 : (\tau_2, \theta_2) \\ \theta_3 = \mathcal{U}(\theta_2\theta_0\tau_1, \text{bool}) \qquad \theta_4 = \mathcal{U}(\theta_3\tau_2, \theta_3\theta_2\tau_1)\end{array}}{\Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : (\theta_4\theta_3\tau_2, \theta_4\theta_3\theta_2\theta_1\theta_0)}\ \textsc{If}$$

$$\frac{\Gamma \vdash e_1 : (\tau_1, \theta_1) \qquad \theta_1\Gamma, x{:}\tau_1 \vdash e_2 : (\tau_2, \theta_2)}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : (\tau_2, \theta_2\theta_1)}\ \textsc{Let}$$

Unification

- “classical” algorithm ([Robinson, 1965])
- many applications (theorem proving, Prolog etc.)
- unifier of two types[7] τ_1 and τ_2: a substitution θ such that

$$\theta(\tau_1) = \theta(\tau_2)$$

- unfication problem: given τ_1 and τ_2, determine a *unifier* for them, if it exists

[7] in other areas: formulas, terms . . .

Unification

- "classical" algorithm ([Robinson, 1965])
- many applications (theorem proving, Prolog etc.)
- unifier of two types[7] τ_1 and τ_2: a substitution θ such that

$$\theta(\tau_1) = \theta(\tau_2)$$

- better formulation of unfication problem: given τ_1 and τ_2, determine the best = most general unifier for them (if they are unifiable).

[7] in other areas: formulas, terms ...

Unification algorithm for underlying types

$$
\begin{array}{rcl}
\mathcal{U}(\text{int}, \text{int})) & = & \textit{id} \\
\mathcal{U}(\text{bool}, \text{bool})) & = & \textit{id} \\
\mathcal{U}(\tau_1 \to \tau_2, \tau_1' \to \tau_2') & = & \begin{array}{ll} \texttt{let} & \theta_1 = \mathcal{U}(\tau_1, \tau_1') \\ & \theta_2 = \mathcal{U}(\theta_1 \tau_2, \theta_1 \tau_2') \\ \texttt{in} & \theta_2 \circ \theta_1 \end{array} \\
\mathcal{U}(\tau, \alpha) & = & \begin{cases} [\alpha \mapsto \tau] & \text{if } \alpha \text{ does not occur in } \tau \\ & \text{or if } \alpha = \tau \\ \text{fail} & \text{else} \end{cases} \\
\mathcal{U}(\alpha, \tau) & = & \text{symmetrically} \\
\mathcal{U}(\tau_1, \tau_2) & = & \text{fail} \quad \text{in all other cases}
\end{array}
$$

Type inference algorithm

- formulated here as rule system
- immediate correspondence to a recursive function:

$$\mathcal{W}(\Gamma, e) = (\tau, \theta)$$

instead of

$$\Gamma \vdash e : (\tau, \theta)$$

“Classic” type inference

- we did not look at the *full* well-known Hindley-Damas-Milner type inference algorithm
- missing here: polymorphic let
- monomoprhic let: “almost useless” polymorphism
- Note the fine line
  - polymorphic let: yes
  - polymorphic functions as function arguments: no!

the classical type “inference” algo

- higher-order functions,
- polymorphic functions,
- but no “higher-order polymorphic functions”

- dropping the last restriction: type inference undecidable
- no type variables in the underlying type system (the “specification”), the type inference algo does
- types (with variables) and *type schemes* $\forall\alpha.\tau$

Extension to CFA

- technical complications due to annotated types $\tau_1 \overset{\varphi}{\to} \tau_2$
- problem: classical unification works with "plain" syntax[8]
- here in contrast, syntactically different (augmented) types considered equal, e.g.:

"UCAI"

$$\tau_1 \overset{\varphi_1 \cup \varphi_2}{\to} \tau_2 = \tau_1 \overset{\varphi_2 \cup \varphi_1}{\to} \tau_2$$

- 2 possible ways out
  - look at the literature for more general unification
  - reduce the problem to classic unification $\mathcal{U}_{\text{UL}}$
    - simple types
    - constraints

[8]Free term algebra.

Simple types and annotations

Goal

"make unification work again"

- simple types: annotation variables instead of annotations

$\Rightarrow$ freely generated type syntax

$\hat{\tau}$	$\in$	**SType**	simple types
α	$\in$	**TVar**	type variables
β	$\in$	**AVar**	annotation variables
φ	$\in$	**SAnn**	simple annotations

Grammar

$$\begin{array}{rcl} \hat{\tau} & ::= & \text{int} \mid \text{bool} \mid \hat{\tau} \xrightarrow{\beta} \hat{\tau} \mid \alpha \\ \alpha & ::= & \texttt{'a} \mid \texttt{'b} \ldots \\ \beta & ::= & \texttt{'1} \mid \texttt{'2} \ldots \\ \varphi & ::= & \{\pi\} \mid \varphi \cup \varphi \mid \emptyset \mid \beta \end{array}$$

Substitutions

- obvious restriction: don't mix up type variables and annotation variables

Simple substitutions

"simple" substitutions map

- type var's $\mapsto$ *simple* types
- annotation var's $\mapsto$ annotation var's

$\Rightarrow$ straightforward generalization of the unification algo

Unification algorithm for simple types $\mathcal{U}_{\mathsf{CFA}}$

$$
\begin{array}{rcl}
\mathcal{U}(\mathsf{int}, \mathsf{int}) & = & \mathit{id} \\
\mathcal{U}(\mathsf{bool}, \mathsf{bool}) & = & \mathit{id} \\
\mathcal{U}(\hat{\tau}_1 \xrightarrow{\beta} \hat{\tau}_2, \hat{\tau}_1' \xrightarrow{\beta'} \hat{\tau}_2') & = & \begin{array}[t]{ll} \mathtt{let} & \theta_0 = [\beta' \mapsto \beta] \\ & \theta_1 = \mathcal{U}(\theta_0 \hat{\tau}_1, \theta_0 \hat{\tau}_1') \\ & \theta_2 = \mathcal{U}(\theta_1(\theta_0 \hat{\tau}_2), \theta_1(\theta_0 \hat{\tau}_2')) \\ \mathtt{in} & \theta_2 \circ \theta_1 \circ \theta_0 \end{array} \\
\mathcal{U}(\hat{\tau}, \alpha) & = & \left\{ \begin{array}{ll} [\alpha \mapsto \hat{\tau}] & \text{if } \alpha \text{ does not occur in } \hat{\tau} \\ & \text{or if } \alpha = \hat{\tau} \\ \text{fail} & \text{else} \end{array} \right. \\
\mathcal{U}(\alpha, \hat{\tau}) & = & \text{symmetrically} \\
\mathcal{U}(\hat{\tau}_1, \hat{\tau}_2) & = & \text{fail} \quad \text{in all other cases}
\end{array}
$$

Note: Unification will never fail on annotation variables β

Properties

Theorem (MGU)

- *If* $\mathcal{U}_{\mathsf{CFA}}(\hat{\tau}_1, \hat{\tau}_2) = \theta$, *then* θ *is a simple substitution such that* $\theta\hat{\tau}_1 = \theta\hat{\tau}_2$.

Properties

Theorem (MGU)

- *If* $\mathcal{U}_{\text{CFA}}(\hat{\tau}_1, \hat{\tau}_2) = \theta$, *then* θ *is a simple substitution such that* $\theta\hat{\tau}_1 = \theta\hat{\tau}_2$.
- *If there is a substitution* θ'' *s.t.* $\theta''\hat{\tau}_1 = \theta''\hat{\tau}_2$, *then there exists* θ' *s.t.* $\mathcal{U}_{\text{CFA}}(\hat{\tau}_1, \hat{\tau}_2) = \theta$ *and*

$$\theta'' = \theta' \circ \theta \ .$$

Constraints

- needed: connection between types and simple types (containing annotation var's)$\Rightarrow$

constraints on ann. var's

$$\beta \supseteq \varphi$$

- note: left-hand side uniformely a variable!
- β: annotation variable, φ: simple annotation
- C: set of $\supseteq$-constraints.
- θC: pointwise application of substitution θ to all constraints in C (i.e., for both sides)

Constraint solving

- splitting substitutions
  1. (ground) type substitution: substitution defined on type variables only and maps to $\hat{\textbf{Type}}$ (no vars)
  2. (ground) annotation substitution: substitution defined on annotation variables, only, and maps to **Ann** (i.e., no vars)
- a substitution covers a "piece of syntax" ($\hat{\Gamma}$, $\hat{\tau}$ ...) when defined on all occurring variables

Solution of a constraint set

a ground annotation substitution θ_A solves C

$$\theta_A \models C$$

- if it covers C and
- for all $\beta \supseteq \varphi$ in C: $\theta_A \beta \supseteq \theta_A \varphi$ is true

- ground substitution as before ⇒ ground validation

Algorithm once more (with constraints)

- judgments now with additional constraints as result

$$\hat{\Gamma} \vdash e : (\hat{\tau}, \theta, C)$$

Interpretation

Under the assumptions $\hat{\Gamma}$, program e possesses type $\hat{\tau}$, imposes the restrictions "embodied" by θ on the type variables, and additionally the annotation variables must adhere to the constraints in C.

- Note
  - type variables/unification constraints solved on-the-fly
  - annotation constraints: postponed
- also possible: postpone unification, as well

Algorithm with constraints (abstraction)

So far:

$$\frac{\alpha \text{ fresh} \qquad \Gamma, x : \alpha \vdash e_0 : (\tau_0, \theta_0)}{\Gamma \vdash \ \texttt{fn}_\pi x \Rightarrow e_0 : ((\theta_0 \alpha) \to \tau_0, \theta_0)} \text{ FN}$$

Algorithm with constraints (abstraction)

$$\frac{\alpha \text{ fresh} \qquad \beta \text{ fresh} \qquad \hat{\Gamma}, x{:}\alpha \vdash e_0 : (\hat{\tau}_0, \theta_0, C_0)}{\hat{\Gamma} \vdash \texttt{fn}_\pi x \Rightarrow e_0 : ((\theta_0\alpha) \xrightarrow{\beta} \tau_0, \theta_0, C')} \text{ FN}$$

Algorithm with constraints (abstraction)

$$\frac{\alpha \text{ fresh} \qquad \beta \text{ fresh} \qquad \hat{\Gamma}, x{:}\alpha \vdash e_0 : (\hat{\tau}_0, \theta_0, C_0)}{\hat{\Gamma} \vdash \mathtt{fn}_\pi x \Rightarrow e_0 : ((\theta_0 \alpha) \xrightarrow{\beta} \tau_0, \theta_0, C_0 \cup \{\beta \supseteq \{\pi\}\})} \text{FN}$$

Algorithm with constraints: application

So far:

$$\frac{\hat{\Gamma} \vdash e_1 : (\hat{\tau}_1, \theta_1) \quad \theta_1\hat{\Gamma} \vdash e_2 : (\hat{\tau}_2, \theta_2) \quad \alpha \text{ fresh} \qquad \theta_3 = \mathcal{U}(\theta_2\hat{\tau}_1, \hat{\tau}_2 \to \alpha)}{\hat{\Gamma} \vdash e_1\ e_2 : (\theta_3\alpha, \theta_3 \circ \theta_2 \circ \theta_1)} \textsc{App}$$

Algorithm with constraints: application

$$\frac{\hat{\Gamma} \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \quad \theta_1 \hat{\Gamma} \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2) \quad \alpha, \beta \text{ fresh} \qquad \theta_3 = \mathcal{U}(\theta_2 \hat{\tau}_1, \hat{\tau}_2 \xrightarrow{\beta} \alpha)}{\hat{\Gamma} \vdash e_1\ e_2 : (\theta_3 \alpha, \theta_3 \circ \theta_2 \circ \theta_1, \ldots ? \ldots)} \text{APP}$$

Algorithm with constraints: application

$$\frac{\hat{\Gamma} \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \quad \theta_1 \hat{\Gamma} \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2) \quad \alpha, \beta \text{ fresh} \qquad \theta_3 = \mathcal{U}(\theta_2 \hat{\tau}_1, \hat{\tau}_2 \xrightarrow{\beta} \alpha)}{\hat{\Gamma} \vdash e_1\ e_2 : (\theta_3 \alpha, \theta_3 \circ \theta_2 \circ \theta_1, \theta_3(\theta_2 C_1) \cup \theta_3 C_2)} \text{App}$$

$$\frac{}{\hat{\Gamma} \vdash c : (\tau_c, id, \emptyset)}\ \textsc{Const} \qquad \frac{}{\hat{\Gamma} \vdash x : (\hat{\Gamma}(x), id, \emptyset)}\ \textsc{Var}$$

$$\frac{\alpha, \beta \text{ fresh} \qquad \hat{\Gamma}, x{:}\alpha \vdash e : (\hat{\tau}, \theta, C)}{\hat{\Gamma} \vdash \underset{\pi}{\texttt{fn}}\ x \Rightarrow e : ((\theta\alpha) \xrightarrow{\theta\beta} \hat{\tau}, \theta, C \cup \{\beta \supseteq \{\pi\}\})}\ \textsc{Fn}$$

$$\frac{\alpha, \alpha_0, \beta_0 \text{ fresh} \quad \hat{\Gamma}, f{:}\alpha \xrightarrow{\beta_0} \alpha_0, x{:}\alpha \vdash e_0 : (\hat{\tau}_0, \theta_0, C_0) \quad \theta_1 = \mathcal{U}(\hat{\tau}_0, \theta_0\alpha_0)}{\hat{\Gamma} \vdash \texttt{fun}_\pi x\ f \Rightarrow e_0 : (\theta_1\theta_0\alpha) \xrightarrow{\theta_1\theta_0\beta_0} \theta_1\hat{\tau}_0, \theta_1 \circ \theta_0, (\theta_1 C_0) \cup \{\theta_1\theta_0\beta_0 \supseteq \{\pi\}\})}\ \textsc{Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \quad \theta_1\hat{\Gamma} \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2) \quad \alpha, \beta \text{ fresh} \qquad \theta_3 = \mathcal{U}(\theta_2\hat{\tau}_1, \hat{\tau}_2 \xrightarrow{\beta} \alpha)}{\hat{\Gamma} \vdash e_1\ e_2 : (\theta_3\alpha, \theta_3 \circ \theta_2 \circ \theta_1, \theta_3\theta_2 C_1 \cup \theta_3 C_2)}\ \textsc{App}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_0 : (\hat{\tau}_0, \theta_0, C_0) \qquad \theta_0\Gamma \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \qquad \theta_1\theta_0\Gamma \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2) \\ \theta_3 = \mathcal{U}(\theta_2\theta_1\hat{\tau}_0, \text{bool}) \qquad \theta_4 = \mathcal{U}(\theta_3\hat{\tau}_2, \theta_3\theta_2\hat{\tau}_1) \qquad \theta'_4 = \theta_4\theta_3\theta_2\theta_1\theta_0\end{array}}{\Gamma \vdash \texttt{if}\, e_0 \,\texttt{then}\, e_1 \,\texttt{else}\, e_2 : (\theta_4\theta_3\hat{\tau}_2, \theta'_4, \theta_4\theta_3\theta_2\theta_1 C_0 \cup \theta_4\theta_3\theta_2 C_1 \cup \theta_4\theta_3 C_2)}\ \text{IF}$$

$$\frac{\Gamma \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \qquad \theta_1\Gamma, x{:}\hat{\tau}_1 \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2)}{\Gamma \vdash \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 : (\hat{\tau}_2, \theta_2\theta_1, \theta_2 C_1 \cup C_2)}\ \text{LET}$$

$$\frac{\begin{array}{c}\Gamma \vdash e_1 : (\hat{\tau}_1, \theta_1, C_1) \qquad \theta_2\Gamma \vdash e_2 : (\hat{\tau}_2, \theta_2, C_2) \\ \theta_3 = \mathcal{U}(\theta_2\hat{\tau}_1, \hat{\tau}^1_{\texttt{op}}) \qquad \theta_4 = \mathcal{U}(\theta_3\hat{\tau}_2, \hat{\tau}^2_{\texttt{op}})\end{array}}{\Gamma \vdash e_1 \,\texttt{op}\, e_2 : (\hat{\tau}_{\texttt{op}}, \theta_4\theta_3\theta_2\theta_1, \theta_4\theta_3\theta_2 C_1 \cup \theta_3\theta_3 C_2)}\ \text{OP}$$

Side effect analysis

- so far (for the type/effect part): pure functional language
- now: add states and *side-effects*

⟹ adding *references*

Goal

Determine which references are accessed (creation, read or write), where references are represented by the point of their creation.

- note: in this section:
  - we don't aim at type inference / algorithmic formulation
  - no type variables etc.

Adding side effects

- Extending the syntax
- Be careful about the $:=$ (cf. while-language!!)

$$e \quad ::= \quad \ldots \mid \texttt{new}_\pi x := e_1 \;\texttt{in}\; e_2 \mid\, !x \mid x := e \mid e; e$$

Semantics

- adding mutable "state" to the "configurations"
- $\varsigma \in \textbf{Store} = \textbf{Loc} \to \textbf{Val}$
- Locations/references: ξ

Operational rules

$$\frac{\vdash \langle e_1, \varsigma_2\rangle \rightarrow \langle v_1, \varsigma_2\rangle \qquad \vdash \langle e_2[v_1/x], \varsigma_2\rangle \rightarrow \langle v_2, \varsigma_3\rangle}{\vdash \langle \texttt{let}\; x = e_1 \;\texttt{in}\; e_2, \varsigma_1\rangle \rightarrow \langle v_2, \varsigma_3\rangle}\ \textsc{Let}$$

$$\frac{\vdash \langle e_1, \varsigma_2\rangle \rightarrow \langle v_1, \varsigma_2\rangle \quad \vdash \langle e_2[\xi/x], \varsigma_2[\xi \mapsto v_1]\rangle \rightarrow \langle v_2, \varsigma_3\rangle \quad \xi \notin dom(\varsigma_2)}{\vdash \langle \texttt{new}_\pi\; x := e_1 \;\texttt{in}\; e_2, \varsigma\rangle \rightarrow \langle v_2, \varsigma_3\rangle}\ \textsc{New}$$

$$\frac{}{\vdash \langle !\xi, \varsigma\rangle \rightarrow \langle \varsigma(\xi), \varsigma\rangle}\ \textsc{Deref}$$

$$\frac{\vdash \langle e, \varsigma_1\rangle \rightarrow \langle v, \varsigma_2\rangle}{\vdash \langle \xi := e, \varsigma_1\rangle \rightarrow \langle v, \varsigma_2[\xi \mapsto v]\rangle}\ \textsc{Assgn}$$

$$\frac{\vdash \langle e_1, \varsigma_1\rangle \rightarrow \langle v_1, \varsigma_2\rangle \qquad \vdash \langle e_2, \varsigma_2\rangle \rightarrow \langle v, \varsigma_3\rangle}{\vdash \langle e_1; e_2, \varsigma_1\rangle \rightarrow \langle v_2, \varsigma_3\rangle}\ \textsc{Seq}$$

Annotated types

$$
\begin{array}{lcll}
\varphi & ::= & \{!\pi\} \mid \{\pi :=\} \mid \{\mathtt{new}\,\pi\} \mid \varphi \cup \varphi \mid \emptyset & \text{annotations/effects} \\
\varpi & ::= & \pi \mid \varpi \cup \varpi \mid \emptyset & \text{sets of program points} \\
\hat{\tau} & ::= & \mathsf{int} \mid \mathsf{bool} \mid \hat{\tau} \xrightarrow{\varphi} \hat{\tau} \mid \mathtt{ref}_{\varpi}(\hat{\tau}) & \text{annotated types}
\end{array}
$$

Side-effect analysis

$$\frac{}{\hat{\Gamma} \vdash c : \hat{\tau}_c :: \emptyset}\ \text{T-Const} \qquad \frac{\hat{\Gamma}(x) = \hat{\tau}}{\hat{\Gamma} \vdash x : \tau :: \emptyset}\ \text{T-Var}$$

$$\frac{\hat{\Gamma}, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2 :: \varphi}{\hat{\Gamma} \vdash \ \mathtt{fn}_\pi x \Rightarrow e : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 :: \emptyset}\ \text{T-Fn}$$

$$\frac{\hat{\Gamma}, f{:}\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2 :: \varphi}{\hat{\Gamma} \vdash \ \mathtt{fun}_f\ x \Rightarrow e : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 :: \emptyset}\ \text{T-Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_2 \xrightarrow{\varphi_0} \hat{\tau} :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash e_1\ e_2 : \hat{\tau} :: \varphi_1 \cup \varphi_2 \cup \varphi_0}\ \text{T-App}$$

Side-effect analysis

$$\frac{\hat{\Gamma} \vdash e_0 : \mathsf{bool} :: \varphi_0 \qquad \hat{\Gamma} \vdash e_1 : \hat{\tau} :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau} :: \varphi_2}{\hat{\Gamma} \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \hat{\tau} :: \varphi_0 \cup \varphi_1 \cup \varphi_2}\ \text{T-IF}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma}, x{:}\hat{\tau}_1 \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \hat{\tau}_2 :: \varphi_1 \cup \varphi_2}\ \text{T-LET}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash e_1; e_2 : \hat{\tau}_2 :: \varphi_1 \cup \varphi_2}\ \text{T-SEQ}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma} \vdash e_1 : \hat{\tau}_2 :: \varphi_2 \qquad \mathtt{op} : \hat{\tau}_1 \times \hat{\tau}_2 \to \hat{\tau}}{\hat{\Gamma} \vdash e_1\ \mathtt{op}\ e_2 : \hat{\tau} :: \varphi_1 \cup \varphi_2}\ \text{T-OP}$$

Side-effect analysis

$$\frac{\hat{\Gamma}(x) = \mathtt{ref}_{\{\pi_1,\ldots,\pi_k\}}(\hat{\tau})}{\hat{\Gamma} \vdash\ !x : \hat{\tau} :: \{!\pi_1, \ldots, !\pi_k\}} \ \textsc{T-Deref}$$

$$\frac{\hat{\Gamma}(x) = \mathtt{ref}_{\{\pi_1,\ldots,\pi_k\}}(\hat{\tau}) \qquad \hat{\Gamma} \vdash e : \hat{\tau} :: \varphi}{\hat{\Gamma} \vdash x := e : \hat{\tau} :: \varphi \cup \{\pi_1 :=, \ldots, \pi_k :=\}} \ \textsc{T-Ass}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma}, x{:}\,\mathtt{ref}_{\pi}(\hat{\tau}_1) \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash \mathtt{new}_{\pi} x := e_1 \ \mathtt{in}\ e_2 : \hat{\tau}_2 :: \varphi_1 \cup \varphi_2 \cup \{\mathtt{new}\,\pi\}} \ \textsc{T-New}$$

$$\frac{\hat{\Gamma} \vdash e : \hat{\tau}' :: \varphi' \qquad \hat{\tau}' \leq \hat{\tau} \qquad \varphi' \subseteq \varphi}{\hat{\Gamma} \vdash e : \hat{\tau} :: \varphi} \ \textsc{T-Sub}$$

Sub-effecting and subtyping

- adding the necessary slack
- subtyping: contra- and co-variant

Sub-effecting and subtyping

$$\frac{}{\hat{\tau} \leq \hat{\tau}} \text{ S-REFL}$$

$$\frac{\hat{\tau}_1' \leq \hat{\tau}_1 \qquad \hat{\tau}_2 \leq \hat{\tau}_2' \qquad \varphi \subseteq \varphi'}{\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 \leq \hat{\tau}_1' \xrightarrow{\varphi'} \hat{\tau}_2'} \text{ S-ARROW}$$

$$\frac{\hat{\tau}_1 \gtrless \hat{\tau}_2 \qquad \varpi_1 \subseteq \varpi_2}{\underset{\varpi_1}{\mathtt{ref}}(\hat{\tau}_1) \leq \underset{\varpi_2}{\mathtt{ref}}(\hat{\tau}_2)} \text{ S-R}$$

Exceptions

- standard type (and/or effect) analysis
- exceptions: leaving/breaking out of the standard control-flow
- exception: “non-standard termination” = effect

Syntax: additonal constructs for exceptions

$$e ::= \ldots \mid \texttt{raise}\, s \mid \texttt{handle}\, s\, \texttt{as}\, e_1\, \texttt{in}\, e_2$$

Annotations and type schemes

$$
\begin{array}{lcll}
\varphi & ::= & \{s\} \mid \varphi \cup \varphi \mid \emptyset \mid \beta & \text{annotations} \\
\hat{\tau} & ::= & \text{int} \mid \text{bool} \mid \hat{\tau} \xrightarrow{\varphi} \hat{\tau} \mid \alpha & \text{annotated types} \\
\hat{\sigma} & ::= & \forall \vec{\zeta}.\hat{\tau} &
\end{array}
$$

Semantics: additional rules

$$\frac{\vdash e_1 \rightarrow \texttt{raise}\, s}{\vdash e_1\ e_2 \rightarrow \texttt{raise}\, s}\ \textsc{App}_1$$

$$\frac{\vdash e_1 \rightarrow (\texttt{fn}_\pi\, x \Rightarrow e_0) \qquad \vdash e_2 \rightarrow \texttt{raise}\, s}{\vdash e_1\ e_2 \rightarrow \texttt{raise}\, s}\ \textsc{App}_2$$

$$\frac{\vdash e_1 \rightarrow (\texttt{fn}_\pi\, x \Rightarrow e_0) \qquad \vdash e_2 \rightarrow v_2 \qquad \vdash e_0[v/v_2] \rightarrow \texttt{raise}\, s}{\vdash e_1\ e_2 \rightarrow \texttt{raise}\, s}\ \textsc{App}_3$$

$$\vdash \texttt{raise}\, s \rightarrow \texttt{raise}\, s \qquad \textsc{Raise}$$

$$\frac{\vdash e_2 \rightarrow v \qquad v_2 \neq \texttt{raise}\, s}{\vdash \texttt{handle}\, s\, \texttt{as}\, e_1\, \texttt{in}\, e_2 \rightarrow v_2}\ \textsc{Handle}_1$$

$$\frac{\vdash e_2 \rightarrow \texttt{raise}\, s \qquad \vdash e_1 \rightarrow v_1}{\vdash \texttt{handle}\, s\, \texttt{as}\, e_1\, \texttt{in}\, e_2 \rightarrow v_1}\ \textsc{Handle}_2$$

Exception analysis

$$\frac{}{\hat{\Gamma} \vdash c : \hat{\tau}_c :: \emptyset}\ \text{T-Const} \qquad \frac{\hat{\Gamma}(x) = \hat{\sigma}}{\hat{\Gamma} \vdash x : \hat{\sigma} :: \emptyset}\ \text{T-Var}$$

$$\frac{\hat{\Gamma}, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2 :: \varphi}{\hat{\Gamma} \vdash \underset{\pi}{\texttt{fn}}\, x \Rightarrow e : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 :: \emptyset}\ \text{T-Fn}$$

$$\frac{\hat{\Gamma}, f{:}\hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2, x{:}\hat{\tau}_1 \vdash e : \hat{\tau}_2 :: \varphi}{\hat{\Gamma} \vdash \underset{\pi}{\texttt{fun}}\, f\ x \Rightarrow e : \hat{\tau}_1 \xrightarrow{\varphi} \hat{\tau}_2 :: \emptyset}\ \text{T-Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_2 \xrightarrow{\varphi_0} \hat{\tau} :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash e_1\ e_2 : \hat{\tau} :: \varphi_1 \cup \varphi_2 \cup \varphi_0}\ \text{T-App}$$

Exception analysis

$$\frac{\hat{\Gamma} \vdash e_0 : \text{bool} :: \varphi_0 \qquad \hat{\Gamma} \vdash e_1 : \hat{\tau} :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau} :: \varphi_2}{\hat{\Gamma} \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \hat{\tau} :: \varphi_0 \cup \varphi_1 \cup \varphi_2}\ \text{T-IF}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\sigma}_1 :: \varphi_1 \qquad \hat{\Gamma}, x{:}\hat{\sigma}_1 \vdash e_2 : \hat{\sigma}_2 :: \varphi_2}{\hat{\Gamma} \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \hat{\sigma}_2 :: \varphi_1 \cup \varphi_2}\ \text{T-LET}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2 :: \varphi_2}{\hat{\Gamma} \vdash e_1 ; e_2 : \hat{\tau}_2 :: \varphi_1 \cup \varphi_2}\ \text{T-SEQ}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1 :: \varphi_1 \qquad \hat{\Gamma} \vdash e_1 : \hat{\tau}_2 :: \varphi_2 \qquad \text{op} : \hat{\tau}_1 \times \hat{\tau}_2 \to \hat{\tau}}{\hat{\Gamma} \vdash e_1\ \text{op}\ e_2 : \hat{\tau} :: \varphi_1 \cup \varphi_2}\ \text{T-OP}$$

Exception analysis

$$\frac{}{\hat{\Gamma} \vdash \texttt{raise}\, \boldsymbol{s} : \hat{\tau} :: \{s\}} \text{ T-RAISE}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \tau :: \varphi_1 \qquad \hat{\Gamma} \vdash e_1 : \tau :: \varphi_2}{\hat{\Gamma} \vdash \texttt{handle}\, \boldsymbol{s}\, \texttt{as}\, e_1\, \texttt{in}\, e_2 : \tau :: \varphi_1 \cup (\varphi_2 \setminus \{s\})} \text{ T-HANDLE}$$

$$\frac{\hat{\Gamma} \vdash e : \hat{\tau}' :: \varphi' \qquad \hat{\tau}' \leq \hat{\tau} \qquad \varphi' \subseteq \varphi}{\hat{\Gamma} \vdash e : \hat{\tau} :: \varphi} \text{ T-SUB}$$

$$\frac{\hat{\Gamma} \vdash e : \hat{\tau} :: \varphi \qquad \vec{\zeta} \text{ do not occur free in } \hat{\tau} \text{ and } \hat{\Gamma}}{\hat{\Gamma} \vdash e : \forall \vec{\zeta}.\, \hat{\tau} :: \varphi} \text{ T-GEN}$$

$$\frac{\hat{\Gamma} \vdash e : \forall \vec{\zeta}.\, \hat{\tau} :: \varphi \qquad \theta \text{ has } dom(\theta) \subseteq \{\zeta_1, \ldots, \zeta_n\}}{\hat{\Gamma} \vdash e : \theta\tau :: \varphi} \text{ T-INST}$$

Memory management and regions

Semantics

$$\frac{o \notin dom(\varsigma(rn)) \qquad n = (rn, o)}{\rho \vdash \langle c \; \texttt{at} \; rn, \varsigma\rangle \rightarrow \langle n, \varsigma[rn \mapsto c]\rangle} \text{ E-CONST}$$

$$\frac{}{\rho \vdash \langle x, \varsigma\rangle \rightarrow \langle \rho(x), \varsigma\rangle} \text{ E-VAR}$$

$$\frac{o \notin dom(\varsigma(rn)) \qquad n = (rn, o)}{\rho \vdash \langle \texttt{fn}_\pi x \Rightarrow ee) \; \texttt{at} \; rn, \varsigma\rangle \rightarrow \langle n, \varsigma[n \mapsto \langle x, ee, \rho\rangle]\rangle} \text{ E-FN}$$

$$\frac{o \notin dom(\varsigma(rn)) \qquad n = (rn, o)}{\rho \vdash \langle \texttt{fun}_\pi f[\vec{\varrho}]x \Rightarrow ee \; \texttt{at} \; rn, \varsigma\rangle \rightarrow \langle n, \varsigma[n \mapsto \langle \vec{\varrho}, x, ee, \rho[f \mapsto n]\rangle]\rangle} \text{ E-FUN}$$

$$\frac{\begin{array}{c} \rho \vdash \langle ee_1, \varsigma_1\rangle \rightarrow \langle n_1, \varsigma_2\rangle \qquad \rho \vdash \langle ee_2, \varsigma_2\rangle \rightarrow \langle v_2, \varsigma_3\rangle \\ n_1 = (rn_1, o_1) \qquad \varsigma_3(n_1) = \langle x, ee_0, \rho_0\rangle \qquad \rho_0[x \mapsto v_2] \vdash \langle ee_0, \rho_3\rangle \rightarrow \langle v_0, \varsigma_4\rangle \end{array}}{\rho \vdash \langle ee_1 \; ee_2, \varsigma_1\rangle \rightarrow \langle v_0, \varsigma_4\rangle}$$

$$\frac{\rho \vdash \langle ee_0, \varsigma_1 \rangle \to \langle n, \varsigma_2 \rangle \qquad n = (rn, o) \qquad \varsigma_2(n) = \text{true} \qquad \rho \vdash \langle ee_1, \varsigma_2 \rangle \to \langle v_1, \varsigma_3 \rangle}{\rho \vdash \langle \texttt{if}\; ee_0 \;\texttt{then}\; ee_1 \;\texttt{else}\; ee_2, \varsigma_1 \rangle \to \langle v_1, \varsigma_3 \rangle} \;\text{E-IF}_1$$

$$\frac{\rho \vdash \langle ee_1, \varsigma_1 \rangle \to \langle v_1, \varsigma_2 \rangle \qquad \rho[x \mapsto v_1] \vdash \langle ee_2, \varsigma_2 \rangle \to \langle v_2, \varsigma_3 \rangle}{\rho \vdash \langle \texttt{let}\; x = ee_1 \;\texttt{in}\; ee_2, \varsigma_1 \rangle \to \langle v_2, \varsigma_3 \rangle} \;\text{E-LET}$$

$$\frac{\begin{array}{c}\rho \vdash \langle ee_1, \varsigma_1 \rangle \to \langle n_1, \varsigma_2 \rangle \qquad \rho \vdash \langle ee_2, \varsigma_2 \rangle \to \langle n_2, \varsigma_3 \rangle \\ n_1 = (rn_1, o_1) \quad n_2 = (rn_2, o_2) \quad n = (rn, o) \qquad o \notin dom(\varsigma_3(rn)) \\ w = \varsigma_3(n_1) \;\mathbf{op}\; \varsigma_3(n_2)\end{array}}{\rho \vdash \langle ee_1 \;\texttt{op}\; ee_2 \;\texttt{at}\; rn, \varsigma_1 \rangle \to \langle n, \varsigma_3[n \mapsto w] \rangle} \;\text{E-OP}$$

Semantics

$$\frac{\rho \vdash \langle ee, \varsigma_1 \rangle \to \langle n', \varsigma_2 \rangle \qquad n' = (rn', o') \qquad \varsigma_2(n') = \langle \vec{\varrho}, x, ee_0, \rho_0 \rangle \qquad n = (rn, o) \qquad o \notin dom(\varsigma_2(rn))}{\rho \vdash \langle ee[\vec{rn}]\ \texttt{at}\ rn, \varsigma_1 \rangle \to \langle n, \varsigma_2[n \mapsto \langle x, ee_0[\vec{rn}/\vec{\varrho}], \rho_0 \rangle] \rangle} \text{ E-PLACE}$$

$$\frac{\vec{rn} \cap dom(\varsigma_1) = \emptyset \qquad \rho \vdash \langle ee[\vec{rn}/\vec{\varrho}], \varsigma_1[\vec{rn} \mapsto \vec{[]}] \rangle}{\rho \vdash \langle \texttt{letregion}\ \vec{\varrho}\ \texttt{in}\ ee, \varsigma_1 \rangle \to \langle v, \varsigma_2 \setminus\setminus \vec{rn} \rangle} \text{ E-REGION}$$

Transformation

$$\frac{}{\hat{\Gamma} \vdash \mathit{cleadstoc}\ \mathtt{at}\ r : \hat{\tau}_c@r :: \{\mathtt{put}\ r\}}\ \text{TT-Const}$$

$$\frac{\hat{\Gamma}(x) = \hat{\sigma}}{\hat{\Gamma} \vdash x\mathit{leadstox} : \hat{\sigma} :: \emptyset}\ \text{TT-Var}$$

$$\frac{\hat{\Gamma}, x{:}\hat{\tau}_1@r_1 \vdash \mathit{eleadstoee} : \hat{\tau}_2@r_2 :: \varphi}{\hat{\Gamma} \vdash\ \mathtt{fn}_\pi x \Rightarrow \mathit{eleadsto}(\mathtt{fn}_\pi x \Rightarrow e)\ \mathtt{at}\ r : (\hat{\tau}_1@r_1 \xrightarrow{\beta.\varphi} \hat{\tau}_2@r_2)@r :: \{\mathtt{put}\ r\}}\ \text{TT-F}$$

$$\frac{\hat{\sigma} = (\forall\vec{\beta}, [\vec{\varrho}].\hat{\tau}) \qquad \vec{\beta}, \vec{\varrho}\ \text{do not occur free in}\ \hat{\Gamma}\ \text{and}\ \varphi \qquad \hat{\Gamma}, f{:}\hat{\sigma}@r \vdash \mathtt{fn}_\pi x \Rightarrow \mathit{eleadsto}\mathtt{fn}_\pi x \Rightarrow e\ \mathtt{at}\ r : \hat{\tau}@r :: \varphi}{\hat{\Gamma} \vdash\ \underset{\pi}{\mathtt{fn}}\ x \Rightarrow \mathit{eleadsto}\mathtt{fun}_\pi f\ x \Rightarrow e\ \mathtt{at}\ r : \hat{\sigma}@r :: \varphi}\ \text{TT-Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1\mathit{leadstoee}_1 : (\hat{\tau}_2@r_2 \xrightarrow{\beta.\varphi} \hat{\tau}@r)@r_1 :: \varphi_1 \qquad \hat{\Gamma} \vdash e_2\mathit{leadstoee}_2 : \hat{\tau}_2@r_2 :: \varphi}{\hat{\Gamma} \vdash e_1\ e_2\mathit{leadstoee}_1\ ee_2 : \hat{\tau}@r :: \varphi_1 \cup \varphi_2 \cup \varphi \cup \{\mathtt{get}\ r_1\}}$$

Behavior and communication

- effects so far: sets of some atomic effects
- no order between effects
- here: effect is "behavior": (temporal) ordering
- especially relevant for concurrency/reactive systems
- $\Rightarrow$

  communication analysis

- vehicle: functional language + communication primitives
- also: new form of semantic definition
- fragment of concurrent ML

Syntax extension

- expression to
  - create channels
  - send and receive over channels
  - start parallel execution
  - sequential composition

$$e \quad ::= \quad \texttt{channel}_\pi \mid \texttt{spawn}\, e_0 \mid \texttt{send}\, e\, \texttt{on}\, e \mid \texttt{receive}\, e_0 \mid e_1 ; e_2 \mid c$$

$$\begin{array}{lll} ch & \in & \textbf{Chan} \qquad \text{channel identifiers} \\ ch & ::= & \texttt{chan1} ::= \texttt{chan2} \mid \ldots \end{array}$$

additional constant $() \in$ **Const** ("unit")

Example

```
let node = fnF f ⇒ fnI inp ⇒ fnO out ⇒
           spawn ((funH h d ⇒ let v = receive inp
                                  in send (f v) on out;
                                     h d) ())

in fun pipe fs ⇒ fnI inp ⇒ fnO out ⇒
       if   isnil fn
       then node (fnX x ⇒ x) inp out
       else let ch = channelC
            in  (node (hd fs) inp ch; piple (tl fs) ch out)
```

Operational semantics

- slight variant in definition
- small-step semantics
- definition with the help of evaluation context
- values

Sequential semantics

$$(\mathtt{fn}_\pi\, x \Rightarrow x)v \to e[v/x] \qquad \textsc{App}$$

$$\mathtt{let}\; x = v \;\mathtt{in}\; e \to e[v/x] \qquad \textsc{Let}$$

$$\frac{v_1 \;\mathbf{op}\; v_2 = v}{v_1 \;\mathtt{op}\; v_2 \to v}\;\textsc{Op}$$

$$\mathtt{fun}_\pi\, f\; x \Rightarrow e \to (\mathtt{fun}_\pi\, f\; x \Rightarrow e)[\mathtt{fun}_\pi\, f\; x \Rightarrow e/f] \qquad \textsc{Rec}$$

$$\mathtt{if}\;\mathsf{true}\;\mathtt{then}\; e_1 \;\mathtt{else}\; e_2 \to e_1 \qquad \textsc{True}$$

$$\mathtt{if}\;\mathsf{false}\;\mathtt{then}\; e_1 \;\mathtt{else}\; e_2 \to e_2 \qquad \textsc{True}$$

$$v; e \to e \qquad \textsc{Seq}$$

Evaluation contexts

- steps on previous slide: not enough
- neither they are meant as "reduction relation":

  Reduce any matching subterm in an expression using $\rightarrow$

Evaluation contexts

- steps on previous slide: not enough
- neither they are meant as "reduction relation":

  Reduce any matching subterm in an expression using $\rightarrow$

$$
\begin{array}{lcl}
E & ::= & [\,] \mid E\ e \mid \texttt{let}\ x = E\ \texttt{in}\ e \\
  & \mid & \texttt{if}\ E\ \texttt{then}\ e\ \texttt{else}\ e \mid E\ \texttt{op}\ e \mid v\ \texttt{op}\ E \\
  & \mid & \texttt{send}\ E\ \texttt{on}\ e \mid \texttt{send}\ v\ \texttt{on}\ E \mid \texttt{receive}\ E \mid E; e
\end{array}
$$

- eval. context: expression with exactly one hole [ ]
- note:
  - positions of values v and real expressions e
  - [ ] does not occur inside binder.

Concurrent semantics

- configurations
  - *PP* finite pool of processes
  - *CP* finite pool of channels

$$
\begin{array}{lcll}
p & \in & \mathbf{Proc} & \text{processes} \\
p & ::= & \texttt{proc1} \mid \texttt{proc2} \mid \ldots & \\
 & & & \\
CP & \in & 2^{\mathbf{Chan}} & \text{finite} \\
PP & \in & \mathbf{Proc} \rightarrow_{fin} \mathbf{Expr} &
\end{array}
$$

- note: $PP(p)$ closed
- notation: $PP[p : e]$ or $PP, p : e$

Concurrent semantics: rules

$$\frac{e_1 \rightarrow e_2}{CP, PP[p : E[e_1]] \rightarrow CP, PP[p : E[e_2]]}\ \textsc{Seq}$$

$$\frac{ch \text{ fresh}}{CP, CP[p : E[\underset{\pi}{\texttt{channel}}]] \rightarrow CP \cup \{ch\}, CP[p : E[ch]]}\ \textsc{Chan}$$

$$\frac{p_0 \text{fresh}}{CP, PP[p_1 : E[\texttt{spawn}\ e_0]] \rightarrow CP, PP[p : E[()]][p_0 : e_0]}\ \textsc{Spawn}$$

$$\frac{p_1 \neq p_2}{\begin{array}{l} CP, PP[p_1 : E_1[\texttt{send}\ v\ \texttt{on}\ ch]][p_2 : E_2[\texttt{receive}\ ch]] \\ \quad \rightarrow CP, PP[p_1 : E_1[()]][p_2 : E_2[v]] \end{array}}\ \textsc{Comm}$$

Annotated types

$$\begin{array}{llll} \hat{\tau} & \in & \textbf{Type}_{\mathrm{CA}} & \text{types} \\ \varphi & \in & \textbf{Ann}_{\mathrm{CA}} & \text{annotations (or behaviors)} \\ r & \in & \textbf{Reg}_{\mathrm{CA}} & \text{region} \\ \hat{\sigma} & \in & \textbf{Scheme}_{\mathrm{CA}} & \text{type scheme} \end{array}$$

types: behaviors:

$$\begin{array}{lll} \varphi & ::= & \beta \mid \Lambda \mid \varphi;\varphi \mid \varphi+\varphi \mid \mathrm{rec}\beta.\varphi \\ & \mid & \hat{\tau}\mathrm{chan}\tau \mid \mathrm{spawn}\varphi \mid r!\hat{\tau} \mid r?\hat{\tau} \end{array}$$

regions

$$r ::= \{\pi\} \mid \varsigma \mid r \cup r \mid \emptyset$$

$\varsigma \in$ **RVar.**

$$\hat{\Gamma} \vdash c : \tau_c \ \&\ \Lambda \quad \textsc{Con}$$

$$\frac{\hat{\Gamma}(x) = \hat{\sigma}}{\hat{\Gamma} \vdash x : \hat{\sigma} \ \&\ \Lambda}\ \textsc{Var}$$

$$\frac{\hat{\Gamma}, x{:}\hat{\tau}_x \vdash e_0 : \hat{\tau}_o \ \&\ \varphi_0}{\hat{\Gamma} \vdash \underset{\pi}{\mathtt{fn}}\, x \Rightarrow e_0 : \hat{\tau}_x \to \hat{\tau}_0 \ \&\ \Lambda}\ \textsc{Fn}$$

$$\frac{\hat{\Gamma}, f{:}\hat{\tau}_x \overset{\varphi_0}{\to} \hat{\tau}_0, x{:}\hat{\tau}_x \vdash e_0 : \hat{\tau}_0 \ \&\ \varphi_0}{\hat{\Gamma} \vdash \underset{\pi}{\mathtt{fun}}\, f\ x \Rightarrow e_0 : \hat{\tau}_x \overset{\varphi_0}{\to} \hat{\tau}_0 \ \&\ \Lambda}\ \textsc{Fun}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_2 \overset{\varphi_0}{\to} \hat{\tau}_0 \ \&\ \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2 \ \&\ \varphi_2}{\hat{\Gamma} \vdash e_1\ e_2 : \hat{\tau}_0 \ \&\ \varphi_1; \varphi_2; \varphi_0}\ \textsc{App}$$

$$\frac{\hat{\Gamma} \vdash e_0 : \text{bool} \mathbin{\&} \varphi_0 \quad \hat{\Gamma} \vdash e_1 : \hat{\tau} \mathbin{\&} \varphi_1 \quad \hat{\Gamma} \vdash e_2 : \hat{\tau} \mathbin{\&} \varphi_2}{\hat{\Gamma} \vdash \texttt{if}\, e_0 \,\texttt{then}\, e_1 \,\texttt{else}\, e_2 : \hat{\tau} \mathbin{\&} \varphi_0; (\varphi_1 + \varphi_2)} \text{ T-IF}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\sigma}_1 \mathbin{\&} \varphi_1 \qquad \hat{\Gamma}, x{:}\hat{\sigma}_1 \vdash e_2 : \hat{\tau}_2 \mathbin{\&} \varphi_2}{\hat{\Gamma} \vdash \texttt{let}\, x = e_1 \,\texttt{in}\, e_2 : \hat{\tau}_2 \mathbin{\&} \varphi_1; \varphi_2} \text{ T-LET}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}^1_{\texttt{op}} \mathbin{\&} \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}^2_{\texttt{op}} \mathbin{\&} \varphi_2}{\hat{\Gamma} \vdash e_1 \,\texttt{op}\, e_2 : \tau_{\texttt{op}} \mathbin{\&} \varphi_1; \varphi_2; \Lambda} \text{ T-OP}$$

$$\hat{\Gamma} \vdash \texttt{channel}_\pi : \hat{\tau}\ \mathsf{chan}\{\pi\}\ \&\ \hat{\tau}\ \mathsf{chan}\{\pi\} \qquad \textsc{T-Chan}$$

$$\frac{\hat{\Gamma} \vdash e_0 : \hat{\tau}_0\ \&\ \varphi_0}{\hat{\Gamma} \vdash \texttt{spawn}\, e_0 : \mathsf{unit}\ \&\ \texttt{spawn}\, \varphi_0}\ \textsc{T-Spawn}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}\ \&\ \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}\ \mathsf{chan} r_2\ \&\ \varphi_2}{\hat{\Gamma} \vdash \texttt{send}\, e_1\, \texttt{on}\, e_2 : \mathsf{unit}\ \&\ \varphi_1; \varphi_2; r_2!\hat{\tau}}\ \textsc{T-Send}$$

$$\frac{\hat{\Gamma} \vdash e_0 : \hat{\tau}\ \mathsf{chan} r_0\ \&\ \varphi_0}{\hat{\Gamma} \vdash \texttt{receive}\, e_0 : \hat{\tau}\ \&\ \varphi_0; r_2?\hat{\tau}}\ \textsc{T-Receive}$$

$$\frac{\hat{\Gamma} \vdash e_1 : \hat{\tau}_1\ \&\ \varphi_1 \qquad \hat{\Gamma} \vdash e_2 : \hat{\tau}_2\ \&\ \varphi_2}{\hat{\Gamma} \vdash e_1; e_2 : \hat{\tau}_2\ \&\ \varphi_1; \varphi_2}\ \textsc{Seq}$$

$$\frac{\hat{\Gamma}(ch) = \hat{\tau}\mathsf{chan} r}{\hat{\Gamma} \vdash ch : \hat{\tau}\mathsf{chan} r\ \&\ \Lambda}\ \textsc{Ch}$$

$$\frac{\hat{\Gamma} \vdash e : \hat{\tau} \;\&\; \varphi \qquad \hat{\tau} \leq \hat{\tau}' \qquad \varphi \sqsubseteq \varphi'}{\hat{\Gamma} \vdash e : \hat{\tau}' \;\&\; \varphi'} \textsc{Sub}$$

$$c\frac{\hat{\Gamma} \vdash e : \hat{\tau} \;\&\; \varphi \qquad \zeta_i \text{ do not occur free in } \hat{\Gamma} \text{ and } \varphi}{\hat{\Gamma} \vdash e : \forall(\zeta_1, \ldots, \zeta_n).\ \hat{\tau} \;\&\; \varsigma}\ \textsc{Gen}$$

$$\frac{\hat{\Gamma} \vdash e : \forall(\zeta_1, \ldots, \zeta_n).\hat{\tau} \;\&\; \varphi \qquad \theta \text{ has } dom(\theta) \subseteq \{\zeta_1, \ldots, \zeta_n\}}{\hat{\Gamma} \vdash e : (\theta\ \hat{\tau}) \;\&\; \varphi}\ \textsc{Ins}$$

the polymorphic let

```
let f = fn x ⇒ x
ins f(f 4);
```

```
let f = fn x ⇒ x
ins f f;
```

```
let f = fn x ⇒ x
ins (f 3); (f true);
```

Ordering on behavior

- preorder on behavior
- algebraic characterization
-

$$\varphi \sqsubseteq \varphi \quad \text{O-Refl} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2 \qquad \varphi_2 \sqsubseteq \varphi_3}{\varphi_1 \sqsubseteq \varphi_3}\ \text{O-Trans}$$

$$\frac{\varphi_1 \sqsubseteq \varphi_2 \qquad \varphi_3 \sqsubseteq \varphi_4}{\varphi_1; \varphi_3 \sqsubseteq \varphi_2; \varphi_4}\ \text{O-Seq} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2 \qquad \varphi_3 \sqsubseteq \varphi_4}{\varphi_1 + \varphi_3 \sqsubseteq \varphi_2 + \varphi_4}\ \text{O-Plus}$$

$$\frac{\varphi_1 \sqsubseteq \varphi_2}{\texttt{spawn}\ \varphi_1 \sqsubseteq \texttt{spawn}\ \varphi_2}\ \text{O-Spawn} \qquad \frac{\varphi_1 \sqsubseteq \varphi_2}{\text{rec}\ \varphi_1 \sqsubseteq \text{rec}\ \varphi_2}\ \text{O-Rec}$$

$$\varphi_1;(\varphi_2;\varphi_3) \equiv (\varphi_1;\varphi_2);\varphi_3)$$

$$(\varphi_1 + \varphi_2);\varphi_3 \equiv \varphi_1;\varphi_3 + \varphi_2;\varphi_3$$

$$\varphi;\Lambda \equiv \varphi \equiv \Lambda;\varphi$$

$$\varphi_1 \sqsubseteq \varphi_1 + \varphi_2 \qquad \varphi_2 \sqsubseteq \varphi_1 + \varphi_2$$

$$\varphi + \varphi \sqsubseteq \varphi \qquad \mathrm{rec}\beta.\varphi \equiv \varphi[\mathrm{rec}.\beta.\varphi/\beta]$$

$$\frac{\hat{\tau} \lesseqgtr \hat{\tau}' \qquad r \subseteq r'}{\hat{\tau}\text{chan } r \sqsubseteq \hat{\tau}'\text{chan} r'}$$

$$\frac{r_1 \sqsubseteq r_2 \qquad \hat{\tau}_1 \leq \hat{\tau}_2}{r_1 ! \hat{\tau}_1 \sqsubseteq r_2 ! \hat{\tau}_2} \qquad \frac{r_1 \sqsubseteq r_2 \qquad \hat{\tau}_2 \leq \hat{\tau}_1}{r_1 ? \hat{\tau}_1 \sqsubseteq t_2 ? \hat{\tau}_2}$$

References I

[Cousot and Cousot, 1977] Cousot, P. and Cousot, R. (1977).
Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *4th POPL, Los Angeles, CA*. ACM.

[Nielson et al., 1999] Nielson, F., Nielson, H.-R., and Hankin, C. L. (1999).
Principles of Program Analysis.
Springer-Verlag.

[Robinson, 1965] Robinson, J. A. (1965).
A machine-oriented logic based on the resolution principle.
Journal of the ACM, 12:23–41.