BEHAVIORAL TYPES FOR HOI CONCURRENT PROGRAMMING

DRAFT 2.1

Luís Caires
Universidade Nova de Lisboa
(based on joint work with Seco, Lourenço, Militão, Aldrich)

NOVALINCS

NOVA Laboratory for
Computer Science and Informatics

BETTY 2016 Limassol Cyprus

typeful programming

- Huge impact on software quality:
  - “Well-typed programs do not go wrong” [Milner]
- Huge impact on programming (as a human activity):
  - types “tame programmers” to write reasonable code
- “Adopted” type systems are purely structural, state oblivious, unable to tackle the challenges of (shared) state concurrency, aliasing, etc (but, see e.g,, Rust).
  - “Well-typed concurrent programs often go wrong” [?]

typing concurrent programming

- A program in a *typed* concurrent programming language should not go wrong !
- Unfortunately, it often does.
- Many relevant contributions from this community:

  Many specific type systems for abstract models (pi, ambients) and properties (deadlock freedom, race absence, fidelity)

  Emphasis on message passing (communication, sessions)

  Results only partially aligned with the expectations and actual problems of mainstream programming technology

  Even harder with general logics (so good to stick to types)

a challenge

- Would it be possible to do for concurrent programming what "classical type theory" did for general programming?
- What could be the core ingredients of a scalable and reasonably general type theory for general concurrency?
- Insights from process algebra and (sub)structural logics suggest that notions of *behavioral types* may help to provide a uniform foundation for typing concurrent programs
- "the essence of concurrency is interference" (also in aliasing)
- In this talk, I discuss a bit this (not so recent) view illustrating with some recent [popl'13, ecoop'14] and ongoing work

a challenge

- Would it be possible to do for concurrent programming what "classical type theory" did for general programming?
- What could be the core ingredients of a scalable and reasonably general type theory for general concurrency?
- Insights from process types and substructural logics (linear and separation) suggest that *behavioural types* may provide the right foundation for taming "real" concurrency
- "the essence of concurrency is interference" (also in aliasing)
- In this talk, I present **behavioural separation types**, building on our recent work [POPL'13,ECOOP'14,ECOOP'16].

programming language

e, f	$::=$	x	*(Variable)*
	$\mid$	$\lambda x.e$	*(Abstraction)*
	$\mid$	$e_1 e_2$	*(Application)*
	$\mid$	$\textbf{let } x = e_1 \textbf{ in } e_2$	*(Definition)*
	$\mid$	$\textbf{ref}$	*(Heap cell alloc)*
	$\mid$	$\textbf{free } e$	*(Heap cell free)*
	$\mid$	$a := e$	*(strong Update)*
	$\mid$	$!a$	*(Dereference)*
	$\mid$	$[l_1 = e_1 \mid \ldots]$	*(Tuple)*
	$\mid$	$e.l$	*(Selection)*
	$\mid$	$\textbf{if } (e_1 == e_2) \textbf{ as } x\ e_3 \textbf{ else } e_4$	*(Match)*
	$\mid$	$\#l(e)$	*(Variant)*
	$\mid$	$\textbf{case } e \textbf{ of } \#l_i(x_i) \rightarrow e_i$	*(Conditional)*
	$\mid$	$\textbf{fork } e$	*(New thread)*
	$\mid$	$\textbf{wait } e$	*(Wait)*
	$\mid$	$\textbf{res } a \textbf{ in } e$	*(resource bundle)*
	$\mid$	$\textbf{open}(a)$	*(enter bundle)*
	$\mid$	$\textbf{close}(a)$	*(leave bundle)*

a queue ADT

$$
\begin{array}{l}
\textsf{\textbf{let}}\ newQueue = \\
\quad \lambda[].\textsf{\textbf{var}}\ hd, tl\ \textsf{\textbf{in}}\ (\\
\qquad hd := \#\texttt{F}(newNode\ \textsf{\textbf{nil}}); tl := \#\texttt{N}; \\
\qquad \textsf{\textbf{res}}\ s\ \textsf{\textbf{in}}\ (\\
\quad\ \ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ \ ]
\end{array}
$$

typical queue configurations

$\textbf{let } newQueue =$

$\quad \lambda[].\textbf{var } hd, tl \textbf{ in } ($

$\qquad hd := \#\texttt{F}(newNode \textbf{ nil}); tl := \#\texttt{N};$

$\qquad \textbf{res } s \textbf{ in } ($

$\quad [$

$\qquad enq = \cdots$

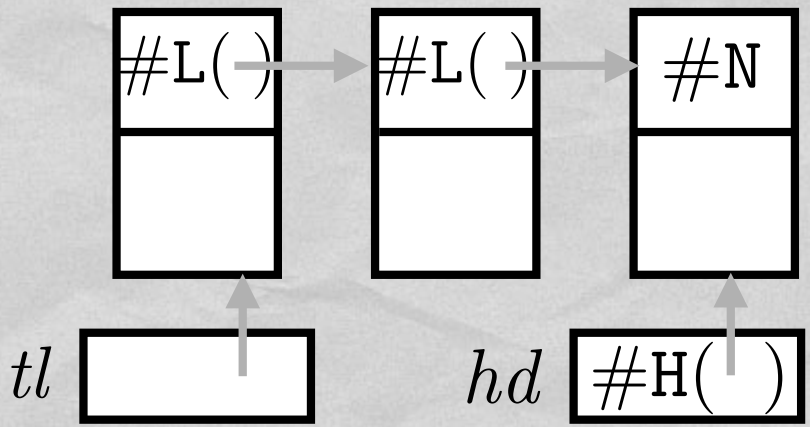
$\qquad |$

$\qquad deq = \cdots$

$\quad]$

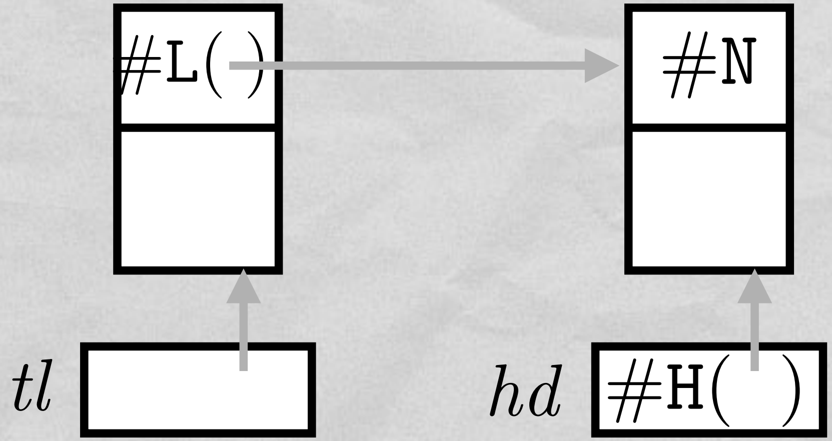
typical queue configurations

$\textbf{let } newQueue =$
$\quad \lambda[].\textbf{var } hd, tl \textbf{ in } ($
$\qquad hd := \#\texttt{F}(newNode \textbf{ nil}); tl := \#\texttt{N};$
$\qquad \textbf{res } s \textbf{ in } ($
$\quad \Big[$
$\qquad enq = \cdots$
$\qquad \Big|$
$\qquad deq = \cdots$
$\quad \Big]$

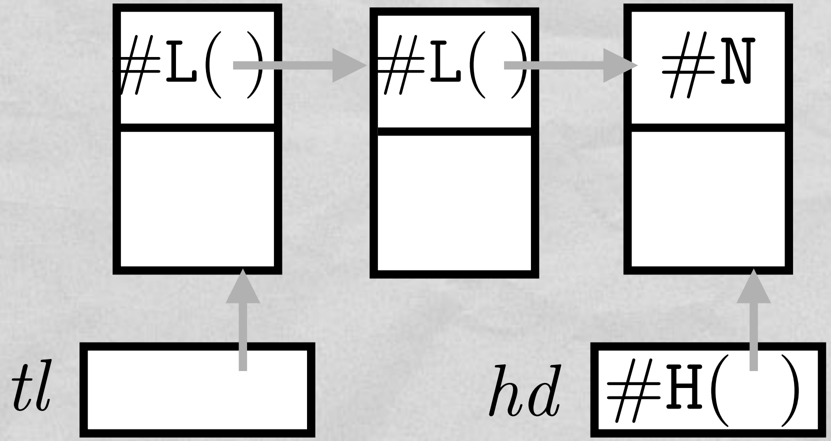


typical queue configurations

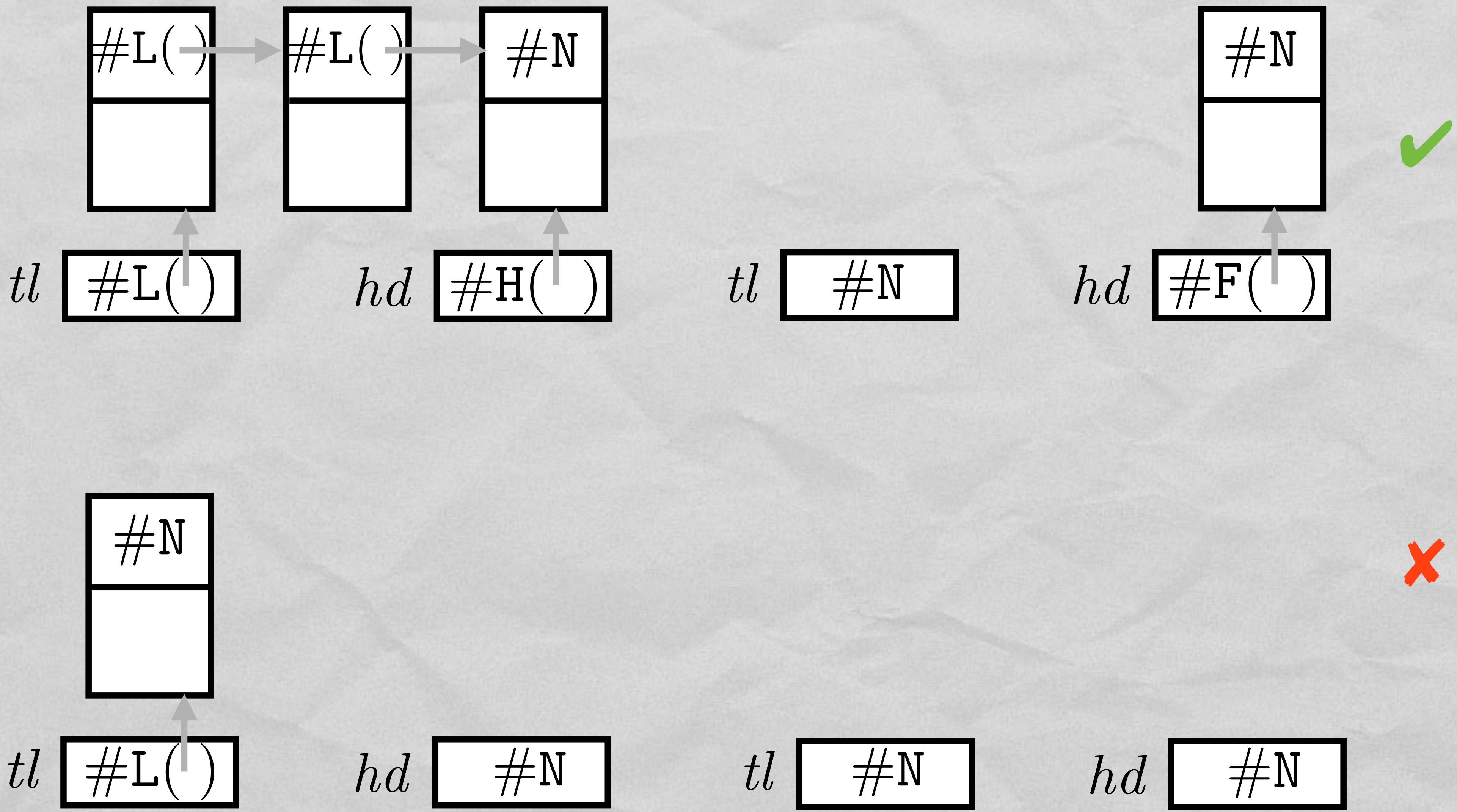
typical queue configurations



typical queue configurations



good / bad queue states



code for queue Node

$$
\begin{aligned}
&\textbf{let } newNode = \lambda[].\textbf{var } nxt := \#\texttt{N} \textbf{ in}\\
&\qquad \textbf{res } n \textbf{ in}\\
&\qquad [\ setNxt = \lambda p.(\textbf{open } n;\\
&\qquad\qquad nx := \#\texttt{L}(p); \textbf{close } n) \mid\\
&\qquad\ \ getNxt = \textbf{open } n;\\
&\qquad\qquad \textbf{let } p = !nxt \textbf{ in } (\textbf{close } n; p) \mid\\
&\qquad\ \ disp = (\textbf{open } n; \textbf{free } nxt; \textbf{close } n)\ ]
\end{aligned}
$$

code for enqueue operation

$$
\begin{array}{l}
enq = \textbf{let}\ n = (newNode\ \textbf{nil})\ \textbf{in} \\
\qquad (\\
\qquad\quad \textbf{open}\ s; \\
\qquad\quad \textbf{case}\ hd\ \textbf{of} \\
\qquad\quad \#\mathtt{H}(hv) \rightarrow (hv.setNxt(n); hd := \#\mathtt{H}(n)) \\
\qquad\quad \#\mathtt{F}(fn) \rightarrow (fn.setNxt(n); hd := \#\mathtt{H}(n); tl := fn); \\
\qquad\quad \textbf{close}\ s \\
\qquad)
\end{array}
$$

code for dequeue operation

$$
\begin{aligned}
deq = \;& \textbf{open}\ s;\\
& \textbf{case}\ hd\ \textbf{of}\\
& \#\texttt{F}(fn) \rightarrow hd := \#\texttt{F}(fn)\\
& \#\texttt{H}(hv) \rightarrow (\\
& \qquad \textbf{case}\ tl.getNxt\ \textbf{of}\\
& \qquad \#\texttt{N} \rightarrow \textbf{nil}\\
& \qquad \#\texttt{L}(tv) \rightarrow (tl.disp;\\
& \qquad\qquad \texttt{if}\ (hv == tv)\ \texttt{as}\ u\\
& \qquad\qquad\ (hd := \#\texttt{F}(u); tl := \#\texttt{N})\\
& \qquad\qquad \texttt{else}\ (hd := \#\texttt{H}(hv); tl := tv));\\
& \textbf{close}\ s
\end{aligned}
$$

client code

$$
\begin{array}{l}
\textbf{let } q = newQueue()\,\textbf{in} \\
\quad \textbf{let } t_1 = \textbf{fork}(\textbf{rec}(X).q.enq; X)\,\textbf{in} \\
\quad\quad \textbf{let } t_2 = \textbf{fork}(\textbf{rec}(X).q.dec; X)\,\textbf{in} \\
\quad\quad\quad (\textbf{wait } t_1; \textbf{wait } t_2)
\end{array}
$$

structural types

$$
\begin{array}{l}
\textbf{let } newQueue = \\
\quad \lambda[].\textbf{var } hd, tl \textbf{ in } (\\
\qquad hd := \#\texttt{F}(newNode\ \textbf{nil}); tl := \#\texttt{N}; \\
\qquad \textbf{res } s \textbf{ in } (\\
\quad\ \ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ \ ]
\end{array}
$$

$newQueue : 0 \rightarrow \{enq : 0, deq : 0\}$ (record type)

behavioral types

$$
\begin{array}{l}
\textbf{let } newQueue = \\
\quad \lambda[].\textbf{var } hd, tl \textbf{ in } (\\
\qquad hd := \#\mathtt{F}(newNode\ \textbf{nil}); tl := \#\mathtt{N}; \\
\qquad \textbf{res } s \textbf{ in } (\\
\quad\ \ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ \ ]
\end{array}
$$

$$newQueue : !(0 \mapsto \mathtt{rec}(X).(enq \;\&\; deq; X))$$

behavioral types

$$
\begin{array}{l}
\textbf{let } newQueue = \\
\quad \lambda[].\textbf{var } hd, tl \textbf{ in } (\\
\qquad hd := \#\mathtt{F}(newNode\ \textbf{nil}); tl := \#\mathtt{N}; \\
\qquad \textbf{res } s \textbf{ in } (\\
\quad\ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ ]
\end{array}
$$

$$newQueue : !(0 \mapsto \mathtt{rec}(X).(enq \ \& \ deq; X))$$

behavioral types

$$
\begin{array}{l}
\textbf{let } \mathit{newQueue} = \\
\quad \lambda[].\textbf{var } hd, tl \textbf{ in } (\\
\qquad hd := \#\mathtt{F}(\mathit{newNode}\ \textbf{nil}); tl := \#\mathtt{N}; \\
\qquad \textbf{res } s \textbf{ in } (\\
\quad\ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ ]
\end{array}
$$

$$\mathit{newQueue} : !0 \mapsto \mathtt{rec}(X).(enq\ ; X) \mid \mathtt{rec}(X).(deq\ ; X)$$

behavioral types

$$
\begin{array}{l}
\textbf{let } newQueue = \\
\quad \lambda[].\textbf{var } hd, tl \textbf{ in } (\\
\qquad hd := \#\texttt{F}(newNode\ \textbf{nil}); tl := \#\texttt{N}; \\
\qquad \textbf{res } s \textbf{ in } (\\
\quad\ [\\
\qquad enq = \cdots \\
\qquad | \\
\qquad deq = \cdots \\
\quad\ ]
\end{array}
$$

$$newQueue : 0 \mapsto (!enq \mid !deq)$$

behavioral types

$$\textbf{let } newQueue =$$
$$\quad \lambda[].\textbf{var } hd, tl \textbf{ in } ($$
$$\qquad hd := \#\texttt{F}(newNode\ \textbf{nil}); tl := \#\texttt{N};$$
$$\qquad \textbf{res } s \textbf{ in } ($$
$$\quad [$$
$$\qquad enq = \cdots$$
$$\qquad |$$
$$\qquad deq = \cdots$$
$$\quad]$$

$$Single = hd{:}\texttt{rd}(\#\texttt{F}(Node)) \mid tl{:}\texttt{rd}(\#\texttt{N})$$

$$Many = hd{:}\texttt{rd}(\#\texttt{H}(Hv)) \mid tl{:}\texttt{rd}(Tv)$$

$$A = (Single \vee Many)\,;(hd{:}\texttt{var} \mid tl{:}\texttt{var})$$

$$s : \rho(inv(A))$$

type structure

behavioral separation types

$$
\begin{array}{lllr|lr}
T, U & ::= & \mathtt{0} & (\textit{stop}) & T \mapsto V & (\textit{function}) \\
 & | & T \,;\, U & (\textit{sequential}) & T \,|\, U & (\textit{parallel}) \\
 & | & T \,\&\, U & (\textit{intersection}) & !T & (\textit{shared}) \\
 & | & T \vee U & (\textit{union}) & l{:}T & (\textit{field}) \\
 & | & \oplus_{l \in I} \# l{:}T_l & (\textit{alternative}) & \rho(A) & (\textit{bundle}) \\
 & | & \circ T & (\textit{isolated}) & \tau(T) & (\textit{thread}) \\
 & | & \mathtt{rec}(X)T & (\textit{recursion}) & X & (\textit{recursion var})
\end{array}
$$

- types express safe *usage* capabilities from *client perspective*
- enforces abstraction / information hiding

recall the linear λ-calculus

$$\frac{A \,|\, x{:}U \vdash e : T}{A \vdash \lambda x.e : U \mapsto T} \; (VAbs)$$

$$\frac{A \vdash e_1 : U \mapsto T \quad B \vdash e_2 : U}{A \,|\, B \vdash e_1 e_2 : T} \; (App)$$

$$0 \vdash \mathbf{nil} : 0$$

$$A, B ::= \mathbf{0} \,\big|\, x{:}T, A$$

$\boxed{A <: B}$ (A is a subtype of B)

$\boxed{A \vdash e : U}$ (e yields value of type U under B)

adding dynamics (cf. Hoare types)

$\boxed{A <: B}$ (A is a subtype of B)

$\boxed{A \vdash_z e :: B}$ (e types from A to z in B)

type assertions (cf. type environments as "global types"):

$$A, B ::= \mathbf{0} \mid x{:}T \mid A\,\mathbf{;}\,B \mid A|B \mid A \,\&\, B \mid A \vee B \mid !A \mid \circ A$$

(linear) arrow type

$$\frac{A \,|\, x{:}U \vdash_y e :: y{:}T}{A \;\vdash_z \lambda x.e :: z{:}U {\mapsto} T}\;(VAbs)$$

$$\frac{A \vdash_z e_1 :: z{:}U {\mapsto} T \qquad B \vdash_x e_2 :: x{:}U}{A \,|\, B \;\vdash_y e_1 e_2 :: y{:}T}\;(App)$$

- symmetric monoidal closed: $(T, 0, (- \,|\, -), \mapsto)$

structural and frame rules

$$x{:}U \vdash_z x :: z{:}U \ (Id)$$

$$\frac{A \vdash_x e_1 :: B \quad B \vdash_y e_2 :: C}{A \vdash_y \textbf{let } x = e_1 \textbf{ in } e_2 :: C} \ (Let/Cut)$$

$$\frac{A <: A' \quad A' \vdash_x e :: B' \quad B' <: B}{A \vdash_x e :: B} \ (Sub)$$

$$\frac{A \vdash_x e :: B}{A \,|\, C \vdash_x e :: B \,|\, C} \ (Par) \qquad \frac{A \vdash_x e :: B}{A \,;\, C \vdash_x e :: B \,;\, C} \ (Seq)$$

laws for parallel and sequence

$U\,;(V\,;T) <:> (U\,;V)\,;T \quad U\,;0 <:> U \quad 0\,;U <:> U$

$U\,|(V\,|\,T) <:> (U\,|\,V)\,|\,T \quad U\,|\,V <:> V\,|\,U \quad U\,|\,0 <:> U$

$(A\,;C)\,|\,(B\,;D) <: (A\,|\,B)\,;(C\,|\,D)$

(special case) $A\,|\,D <: A\,;\,D$

sample typing judgments

$\boxed{A <: B}$ (A is a subtype of B)

$\boxed{A \vdash_z e :: B}$ (e types from A to z in B)

$(f{:}U \mapsto V \,;\, y{:}U) \mid x{:}U \vdash_z (f\ x) :: z{:}V \,;\, y{:}U$ ✔

$(f{:}U \mapsto V \,;\, y{:}U) \mid x{:}U \vdash_z (f\ y) ::$ ✘

$a{:}\texttt{use} \vdash_z (\lambda x.a := x) :: z{:}\circ U \mapsto \texttt{0} \,;\, a{:}\texttt{rd}(\circ U)$ ✔

field type

$$\frac{A \vdash_x e :: x{:}U}{A \vdash_z [\ldots l = e \ldots] :: z{:}l{:}U}(Field)$$

$$\frac{A \vdash_z e :: z{:}l{:}T}{A \vdash_x e.l :: x{:}T}(Sel)$$

(linear) intersection type

$$\frac{A \vdash_y e :: B \quad A \vdash_y e :: C}{A \vdash_y e :: B \ \& \ C}(And)$$

$$U \& V <: U$$

$$U \& V <: V$$

$$\frac{A \vdash_y e :: B_1 \ \& \ B_2}{A \vdash_y e :: B_i}(AndE)$$

$$U <: U \& U$$

- unlabeled choice (e.g., exporting multiple interfaces)
- examples: $A \vdash [up = e_1 | dn = e_2] :: x{:}(up{:}\mathbf{0} \,\&\, dn{:}\mathbf{0})$

separation types

$$\mathsf{0} \vdash_y v :: \mathsf{0} \; (VStop)$$

$$\frac{A \vdash_y v :: C \qquad B \vdash_y v :: D}{A \,;\, B \vdash_y v :: C \,;\, D} (VSeq)$$

$$\frac{A \vdash_y v{::}C \quad B \vdash_y v{::}D}{A \,|\, B \vdash_y v :: C \,|\, D} (VPar)$$

- Concurrent Kleene Algebra [Hoare]:

$$(T, (- \,\&\, -), (- \,|\, -), (- \,;\, -), \mathsf{0})$$

2014 update on CKA

Developments in Concurrent Kleene Algebra

Tony Hoare[1], Stephan van Staden[2], Bernhard Möller[3], Georg Struth[4], Jules Villard[5], Huibiao Zhu[6], and Peter O'Hearn[7]

[1] Microsoft Research, Cambridge, United Kingdom
[2] ETH Zurich, Switzerland
[3] Institut für Informatik, Universität Augsburg, Germany
[4] Department of Computer Science, The University of Sheffield, United Kingdom
[5] Department of Computing, Imperial College London, United Kingdom
[6] Software Engineering Institute, East China Normal University, China
[7] Facebook, United Kingdom

(linear) labeled sum type

$$\frac{A \vdash_y e_c :: y : \oplus_{l \in I} \#l{:}T_l \quad x_i{:}T_i \mid B \vdash_z e_i :: C}{A \mid B \vdash_z \textbf{case } e_c \textbf{ of } \#l(x) \rightarrow e :: C} \ (Case)$$

$$\frac{A \vdash_z e :: z{:}T_i}{A \vdash_z \#l_i(e) :: z{:} \oplus_{l \in I} \#l{:}T_l} \ (Option)$$

- labeled choice (cf. variant type)

(linear) union type

$$\frac{A \vdash_z e :: C \quad B \vdash e :: C}{A \vee B \vdash_z e :: C} \; (UnionCase)$$

$$\frac{A \vdash_z e :: z{:}T_i}{A \vdash_z e :: z{:} \vee_{l \in I} T_l} \; (InUnion)$$

- unlabeled union (e.g., exporting some interface)
- examples: $s : \texttt{rec}(X).(empty?{:}\#\texttt{T}\,\texttt{;}\, push \vee empty?{:}\#\texttt{F}\,\texttt{;}(pop \,\&\, push)\,\texttt{;}\, X)$

types for sharing and isolation

$$\frac{!A_1 \,|\, \ldots \,|\, !A_n \vdash_x v{::}B}{!A_1 \,|\, \ldots \,|\, !A_n \vdash_x v{::}!B}\ (VShr)$$

$$\frac{\circ A_1 \,|\, \ldots \,|\, \circ A_n \vdash_x e :: B}{\circ A_1 \,|\, \ldots \,|\, \circ A_n \vdash_x e :: \circ B}\ (Iso)$$

- monoidal co-monads: $\circ(-)$ isolated

  $!(-)$ shared

laws for sharing (cf. linear logic !)

$$!U <: U$$

$$!U <: !!U$$

$$0 <: !0$$

$$!U \mid !V <: !(U \mid V)$$

$$!U <: 0$$

$$!U <: !U \mid !U$$

reference type

$\vdash_x \textbf{ref} :: x{:}\texttt{var}\ (Ref)$

$$\frac{A \vdash_z e :: x{:}\texttt{var}}{A \vdash_x \textbf{free}\ e :: x{:}0}\ (Free)$$

$\texttt{var} <: \texttt{use}\,;\texttt{var}$

$\texttt{use} <: \texttt{use}\,;\texttt{use}$

$\texttt{use} <: \texttt{wr}(U)\,;\texttt{rd}(U)$

$\texttt{wr}(0) <: 0 \quad \texttt{rd}(0) <: 0$

$\texttt{rd}(U;V) <: \texttt{rd}(U);\texttt{rd}(V)$

$\texttt{rd}(U \mid V) <: \texttt{rd}(U) \mid \texttt{rd}(V)$

$\texttt{rd}(!U) <: !\texttt{rd}(!U)$

$\texttt{rd}(\circ U)\,;\texttt{var} <: \circ(\texttt{rd}(\circ U)\,;\texttt{var})$

reference type

$$a{:}\mathtt{rd}(U) \vdash_x\ !a :: x{:}U \ (RdVB)$$

$$a{:}\mathtt{rd}(U); \mathtt{use} \vdash_x\ !a :: x{:}U \mid a{:}\mathtt{use} \ (RdVF)$$

$$\frac{A \vdash_z v :: z{:}\circ U \mid a{:}\mathtt{wr}(\circ U)}{A \vdash_z a := v :: \mathsf{0}} \ (WrVF)$$

$$\frac{A \vdash_z v :: z{:}U \mid a{:}\mathtt{use}}{A \vdash_z a := v :: a{:}\mathtt{rd}(U)} \ (WrVB)$$

resource bundle types

- region type assigns bundle a *rely-guarantee* protocol R

  $r : \rho(R)$

- our rely-guarantee protocol are given by:

  $$R ::= \mathtt{0} \mid \{A\}\{B\}; R \mid \{B\}; R \mid R \vee R \mid \mathtt{rec}(X)R \mid X$$

- specific sub-typing laws:

  binary projection op: $\bowtie$

  $\rho(A \vee B)$ <: $\rho(A) \vee \rho(B)$

  $\rho(R)$ <: $\rho(R_1) \mid \rho(R_2)$ $\quad$ if $R \bowtie (R_1 \mid R_2)$

- ***projection*** splits R into *compatible* protocols R_1, R_2,... ensuring coordinated progress of several views / aliases

resource bundle types

$$\frac{r{:}\rho(\{A\}\{\mathtt{0}\}) \,|\, C \vdash_x e :: r{:}\rho(\{B\}\{\mathtt{0}\}) \,|\, D}{A \,|\, C \vdash_x \textbf{res } r \textbf{ in } e :: B \,|\, D}$$

$$r{:}\rho(\{A\}\{B\}; R) \vdash_x \textbf{open } r :: A \,|\, r{:}\rho(\{B\}; R)$$

$$B \,|\, r{:}\rho(\{B\}; R) \vdash_x \textbf{close } r :: r{:}\rho(R)$$

- expressing a simpler invariant:

$$inv(A) \triangleq \mathtt{rec}(X).\{A\}\{A\}; X$$

resource bundle types

$$\frac{r{:}\rho(\{A\}\{\mathtt{0}\})\,|\,C \vdash_x e :: r{:}\rho(\{B\}\{\mathtt{0}\})\,|\,D}{A\,|\,C \vdash_x \textbf{res}\ r\ \textbf{in}\ e :: B\,|\,D}$$

$$r{:}\rho(\{A\}\{B\}\,;R) \vdash_x \textbf{open}\ r :: A\,;B\,|\,r{:}\rho(\{B\}\,;R)$$

$$B\,|\,r{:}\rho(\{B\}\,;R) \vdash_x \textbf{close}\ r :: r{:}\rho(R)$$

- expressing a simple invariant:

$$inv(A) \triangleq \mathtt{rec}(X).\{A\}\{A\};X$$

simple resource bundle types

$$\frac{r{:}\rho(A) \,|\, C \vdash_x e :: r{:}\rho(A) \,|\, D}{A \,|\, C \vdash_x \textbf{res}\ r\ \textbf{in}\ e :: A \,|\, D}$$

$$r{:}\rho(A) \vdash_x \textbf{open}\ r :: A \,|\, r{:}\rho(\{A\})$$

$$B \,|\, r{:}\rho(\{A\}) \vdash_x \textbf{close}\ r :: r{:}\rho(A)$$

queue typing

$$\textbf{let } newQueue =$$
$$\quad \lambda[].\textbf{var } hd, tl \textbf{ in } ($$
$$\qquad hd := \#\mathtt{F}(newNode \textbf{ nil}); tl := \#\mathtt{N};$$
$$\qquad \textbf{res } s \textbf{ in } ($$
$$\quad [$$
$$\qquad enq = \cdots$$
$$\qquad |$$
$$\qquad deq = \cdots$$
$$\quad]$$

$$newQueue : 0 \rightarrow (!enc \,|\, !deq)$$

type checking ensures concurrent safety

region type for queue

let $newQueue =$
$\lambda[]$.**var** hd, tl **in** (
$hd := \#\mathtt{F}(newNode$ **nil**$); tl := \#\mathtt{N};$
res s **in** (
$[$
$enq = \cdots$
$|$
$deq = \cdots$
$]$

$$Single = hd{:}\mathtt{rd}(\#\mathtt{F}(Node)) \mid tl{:}\mathtt{rd}(\#\mathtt{N})$$

$$Many = hd{:}\mathtt{rd}(\#\mathtt{H}(Hv)) \mid tl{:}\mathtt{rd}(Tv)$$

$$A = (Single \vee Many)\,;(hd{:}\mathtt{var} \mid tl{:}\mathtt{var})$$

$$s : \rho(inv(A))$$

typing queue nodes

$$
\begin{aligned}
&\textbf{let } newNode = \lambda[].\textbf{var } nxt := \#\mathtt{N} \textbf{ in}\\
&\qquad \textbf{res } n \textbf{ in}\\
&\qquad [\ setNxt = \lambda p.(\textbf{open } n;\\
&\qquad\qquad nx := \#\mathtt{L}(p); \textbf{close } n) \mid\\
&\qquad getNxt = \textbf{open } n;\\
&\qquad\qquad \textbf{let } p = !nxt \textbf{ in } (\textbf{close } n; p) \mid\\
&\qquad disp = (\textbf{open } n; \textbf{free } nxt; \textbf{close } n)\ ]
\end{aligned}
$$

$newNode \triangleq \mathtt{0} \rightarrow \circ Node$

$Node \triangleq Hv \mid Tv$

$Tv \triangleq \mathtt{rec}(X).getNxt{:}\#\mathtt{L}(Tv)\, ;\, disp{:}0 \vee getNxt{:}\#\mathtt{N}\, ;\, X$

$Hv \triangleq setNxt{:}(Tv \mapsto \mathtt{0})$

typing node bundle

$$\textbf{let } newNode = \lambda[].\textbf{var } nxt := \#\texttt{N} \textbf{ in}$$
$$\textbf{res } n \textbf{ in}$$
$$[\ setNxt = \lambda p.(\textbf{open } n;$$
$$nx := \#\texttt{L}(p); \textbf{close } n)\ |$$
$$getNxt = \textbf{open } n;$$
$$\textbf{let } p = !nxt \textbf{ in } (\textbf{close } n; p)\ |$$
$$disp = (\textbf{open } n; \textbf{free } nxt; \textbf{close } n)\ ]$$

$n : \rho\{nxt : \texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\{0\}$

<:

$n : \rho\,\texttt{rec}(X).(\ \{nxt{:}\texttt{rd}(\#\texttt{L}(Tv))\,;\texttt{var}\}\{nxt{:}\texttt{var}\}\,;\{nxt{:}\texttt{var}\}\{0\}$

$\vee\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\,;X\ )$

|

$n : \rho\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\,;\{nxt{:}\texttt{rd}(\#\texttt{L}(Tv))\,;\texttt{var}\}$

typing node bundle

$$\textbf{let } newNode = \lambda[].\textbf{var } nxt := \#\texttt{N} \textbf{ in}$$
$$\textbf{res } n \textbf{ in}$$
$$[\ setNxt = \lambda p.(\textbf{open } n;$$
$$nx := \#\texttt{L}(p); \textbf{close } n) \mid$$
$$getNxt = \textbf{open } n;$$
$$\textbf{let } p = !nxt \textbf{ in } (\textbf{close } n; p) \mid$$
$$disp = (\textbf{open } n; \textbf{free } nxt; \textbf{close } n)\ ]$$

$$\{nxt : \texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\{0\}$$
$$=$$
$$\texttt{rec}(X).(\ \{nxt{:}\texttt{rd}(\#\texttt{L}(Tv))\,;\texttt{var}\}\{nxt{:}\texttt{var}\}\,;\{nxt{:}\texttt{var}\}\{0\}$$
$$\vee\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\,;X\ )$$
$$\bowtie$$
$$\{nxt{:}\texttt{rd}(\#\texttt{N})\,;\texttt{var}\}\,;\{nxt{:}\texttt{rd}(\#\texttt{L}(Tv))\,;\texttt{var}\}$$

(projection)

credits

- separation logics
- spatial logics for concurrency
- session and conversation types
- linear logic (+ connections with session / behavioral types)

Luís Caires, João Costa Seco:
The type discipline of behavioral separation. POPL 2013

Filipe Militão, Jonathan Aldrich, Luís Caires:
Rely-Guarantee Protocols. ECOOP 2014

Filipe Militão, Jonathan Aldrich, Luís Caires: Composing Interfering Abstract Protocols, ECOOP 2016