Komponenten- und Service-orientierte Softwarekonstruktion

Vorlesung 9: Staged Composition Synthesis

Boris Düdder
LS XIV – Software Engineering

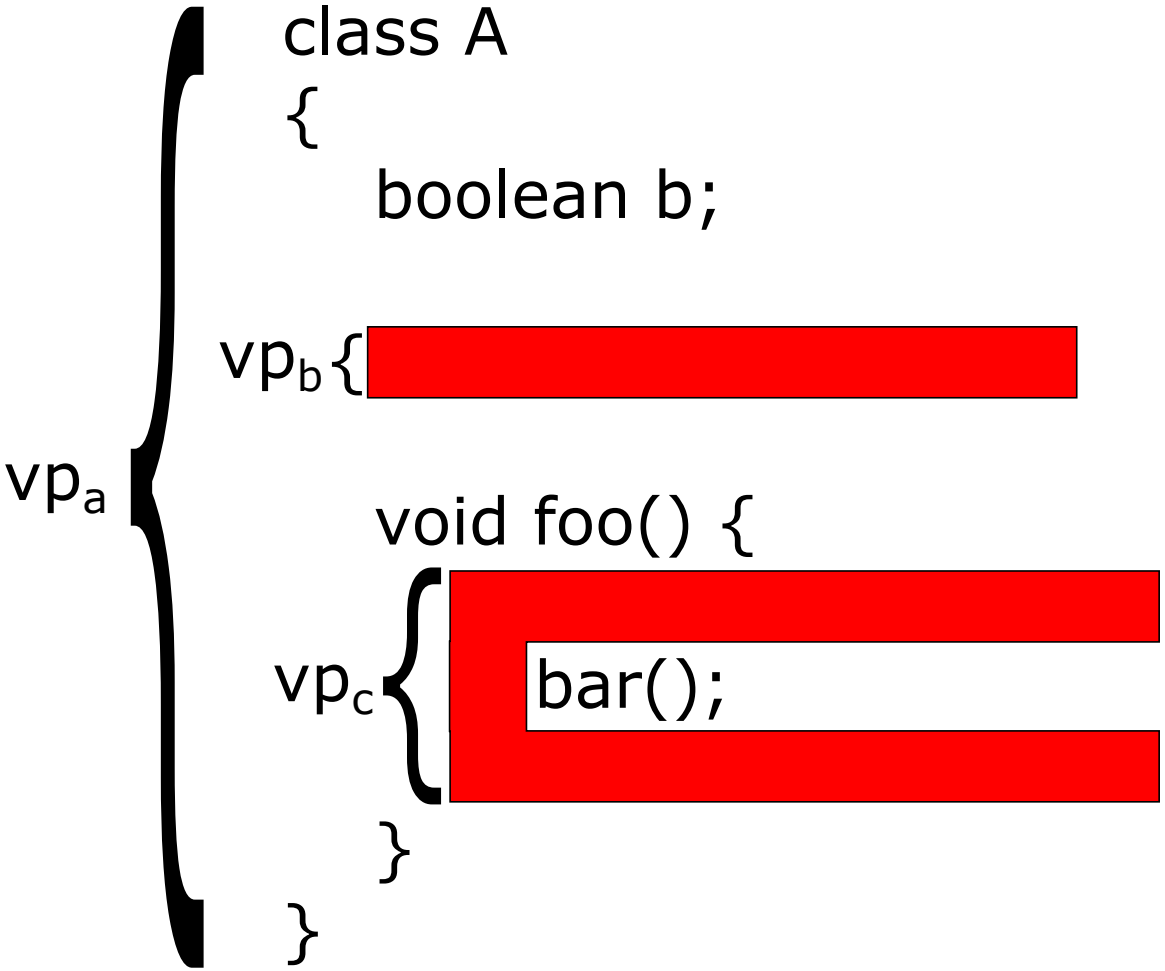
TU Dortmund
Sommersemester 2015

SS 2015

Outline

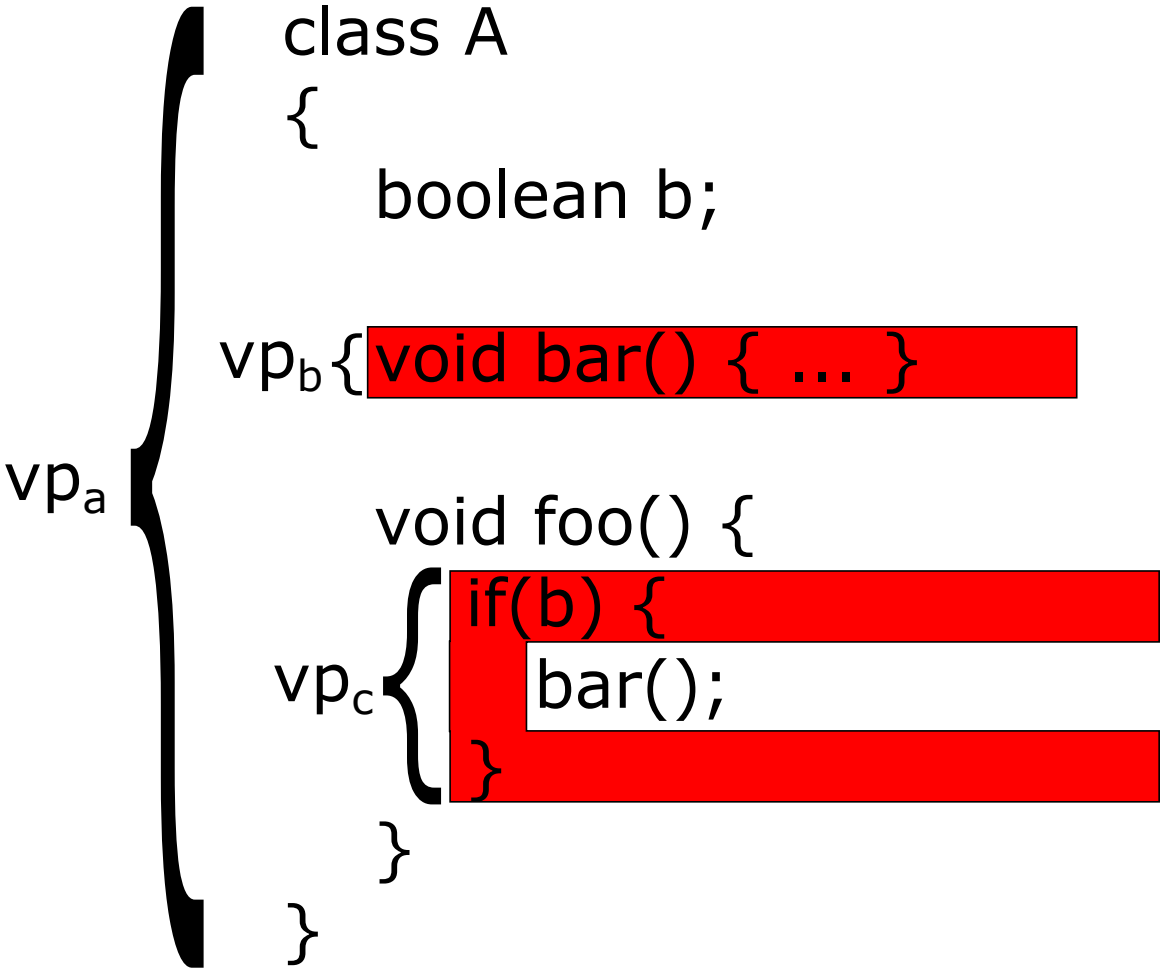
Variation Points

Variation points vp_a, vp_b, and vp_c.



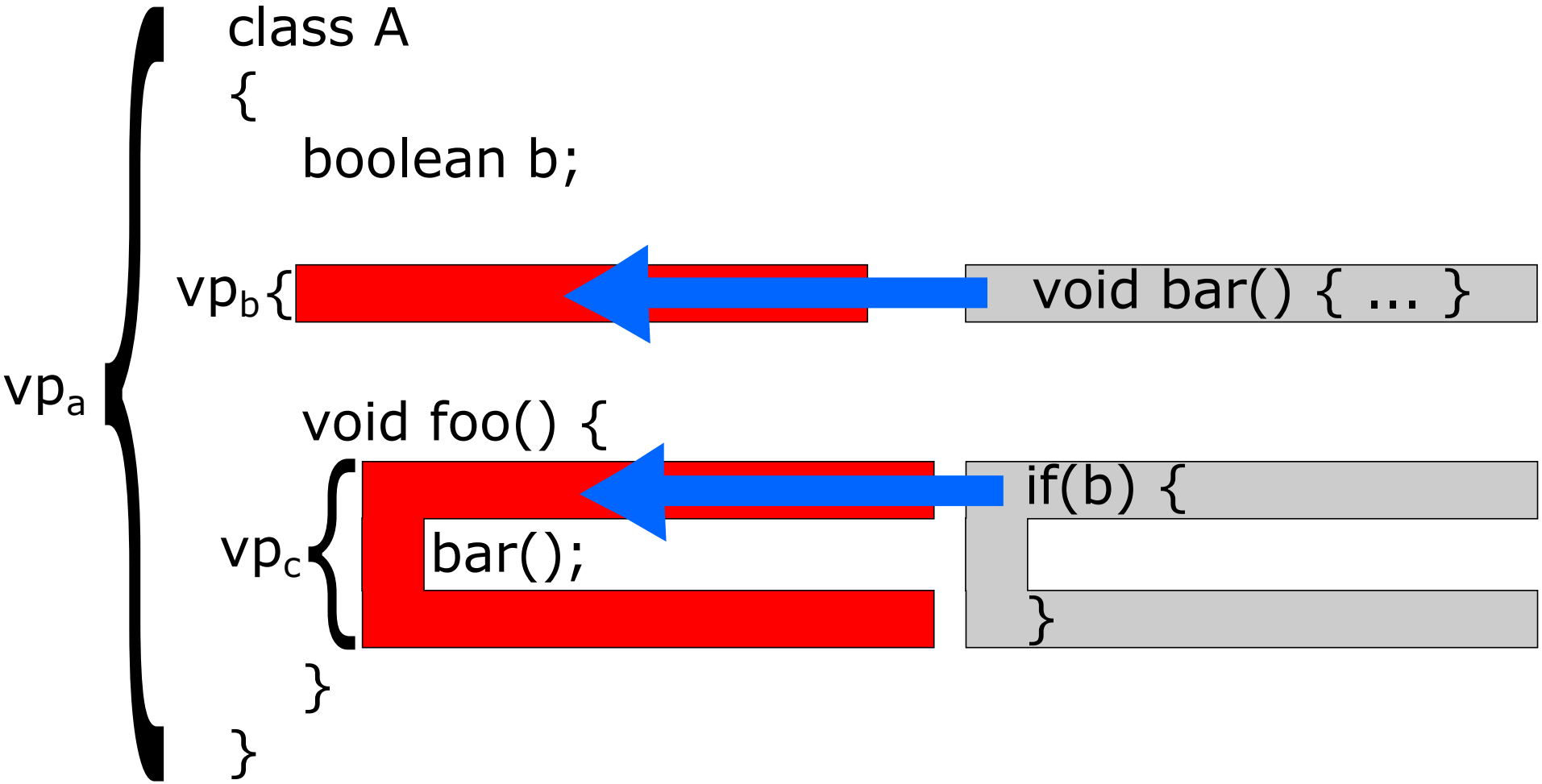
Variation Points

Variation points vp_a, vp_b, and vp_c.



Variation Points

Variation points vp_a, vp_b, and vp_c.



Structured Document Algebra

But wait..., we already know a formalism that uses substitutions.

λ-Calculus (notion of β-reduction)

The notion of β-*reduction* is (reflexive, transitive closure) the relation $\rightarrow_\beta \subseteq \Lambda \times \Lambda$ defined by

$$(\lambda x.M)N \ \rightarrow_\beta \ M[x := N]$$

for all $M, N \in \Lambda$.

Outline

SCS (DMR ESOP 2014) [Düdder et al., 2014]

Goal: Introduce metalanguage L2 *into composition synthesis*

SCS (DMR ESOP 2014) [Düdder et al., 2014]

Goal: Introduce metalanguage L2 *into composition synthesis*

- L1 may be limited, e.g.:
  - L1 might be very low-level
  - L1 might not have a notion of function application

SCS (DMR ESOP 2014) [Düdder et al., 2014]

Goal: Introduce metalanguage L2 *into composition synthesis*

- L1 may be limited, e.g.:
  - L1 might be very low-level
  - L1 might not have a notion of function application
- L2 should contain λ-calculus
  - Special-purpose composition operators
  - Meta-level computation over L1-code
  - Higher-order abstraction

SCS (DMR ESOP 2014) [Düdder et al., 2014]

Goal: Introduce metalanguage L2 *into composition synthesis*

- L1 may be limited, e.g.:
  - L1 might be very low-level
  - L1 might not have a notion of function application
- L2 should contain λ-calculus
  - Special-purpose composition operators
  - Meta-level computation over L1-code
  - Higher-order abstraction
- Challenges
  - How to expose language distinction in types?
  - How to ensure *implementation type correctness*?
  - How to ensure *staged* composition?

Main Ideas in SCS

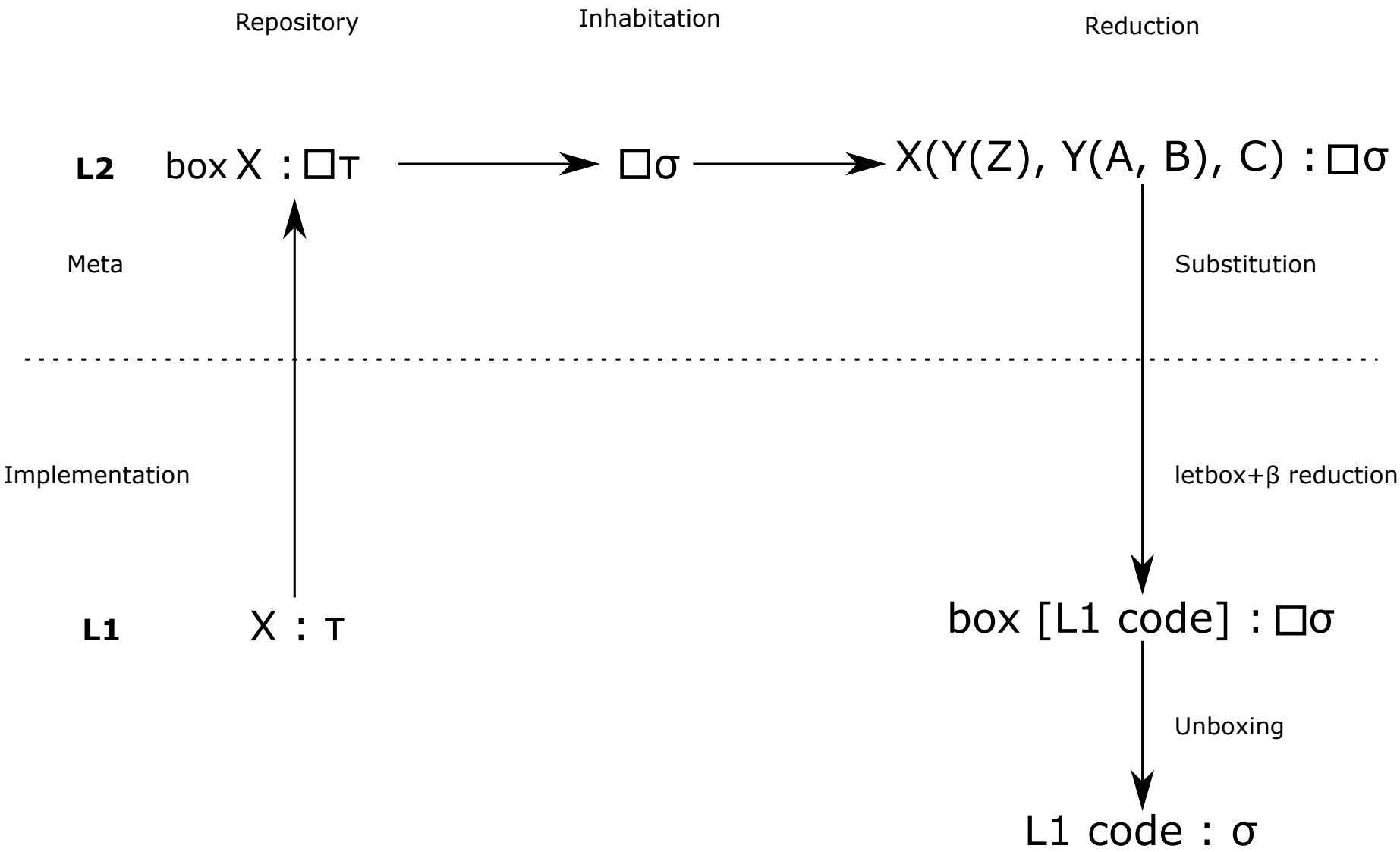
Use modal types $\Box\phi$ (*"code of type ϕ"*) *to expose language distinction to compostion synthesis.*

Main Ideas in SCS

Use modal types $\Box\phi$ *("code of type* ϕ*") to expose language distinction to compostion synthesis.*

- Introduction of modal intersection types
- Davies and Pfenning's calculus [Davies and Pfenning, 2001] $\lambda_e^{\Box,\to}$ as L2 implementation language
- Challenges of staging solved by theory of $\lambda_e^{\Box,\to}$
- Challenge of implementation type correctness solved by conservative extension theorem
- Extension of inhabitation (semi-)algorithm for modal intersection types
- Extension of **(CL)S**-framework and experiments

Overview of SCS



Metalanguage L2

Modal types $\sigma ::= \Box\tau \mid \sigma \to \sigma'$, where τ ranges over L1-types

Metalanguage L2 (adapted from $\lambda_e^{\Box,\to}$) over L1:

$$\frac{}{\Delta; (\Gamma, x : \sigma) \vdash_{\text{L2}} x : \sigma}(\text{var})$$

$$\frac{\Delta; (\Gamma, x : \sigma) \vdash_{\text{L2}} M : \sigma'}{\Delta; \Gamma \vdash_{\text{L2}} \lambda x : \sigma.M : \sigma \to \sigma'}(\to\text{I}) \qquad \frac{\Delta; \Gamma \vdash_{\text{L2}} M_1 : \sigma \to \sigma' \quad \Delta; \Gamma \vdash_{\text{L2}} M_2 : \sigma}{\Delta; \Gamma \vdash_{\text{L2}} (M_1\, M_2) : \sigma'}(\to\text{E})$$

$$\frac{\Delta; \emptyset \vdash_{\text{L1}} T : \tau}{\Delta; \Gamma \vdash_{\text{L2}} \text{box}\ T : \Box\tau}(\Box\text{I}) \qquad \frac{\Delta; \Gamma \vdash_{\text{L2}} M_1 : \Box\tau \quad (\Delta, u : \tau); \Gamma \vdash_{\text{L2}} M_2 : \sigma}{\Delta; \Gamma \vdash_{\text{L2}} \text{letbox}\ u : \tau = M_1\ \text{in}\ M_2 : \sigma}(\Box\text{E})$$

Reduction

$$\text{letbox}\ u = \text{box}\ T\ \text{in}\ M \qquad \longmapsto \qquad M[u := T]$$

Davies and Pfenning: *A Modal Analysis of Staged Computation*, JACM, 2001

Combinatory Logic C2

Semantic types	$\mathfrak{s}$	$::=$	$\mathfrak{b} \mid \mathfrak{s} \to \mathfrak{s}' \mid \mathfrak{s} \cap \mathfrak{s}' \mid \Box\mathfrak{t}$
Interface types	ψ	$::=$	$\sigma \mid \psi \cap \mathfrak{s} \mid \Box\phi \mid \psi \to \psi' \mid \psi \cap \psi'$

Combinatory Logic C2 over C1

$$\mathcal{C} = \{\text{box } X_i : \Box\phi\} \cup \{Y_j : \psi\} \qquad (\Box\phi \text{ by rule of necessitation})$$

Combinatory Logic C2

Semantic types	$\mathfrak{s}$	$::=$	$\mathfrak{b} \mid \mathfrak{s} \to \mathfrak{s}' \mid \mathfrak{s} \cap \mathfrak{s}' \mid \Box\mathfrak{t}$
Interface types	ψ	$::=$	$\sigma \mid \psi \cap \mathfrak{s} \mid \Box\phi \mid \psi \to \psi' \mid \psi \cap \psi'$

Combinatory Logic C2 over C1

$$\mathcal{C} = \{\text{box } X_i : \Box\phi\} \cup \{Y_j : \psi\} \qquad (\Box\phi \text{ by rule of necessitation})$$

Logically, contains modal logic **S4**

$$@ \triangleq \lambda F : \Box(\alpha \to \beta).\lambda X : \Box\alpha.\ \text{letbox } f : \alpha \to \beta = F \text{ in } \text{letbox } x : \alpha = X \text{ in } \text{box } f(x)$$

$$@ : \Box(\alpha \to \beta) \to \Box\alpha \to \Box\beta$$

Combinatory Logic C2

Semantic types	$\mathfrak{s}$	$::=$	$\mathfrak{b} \mid \mathfrak{s} \to \mathfrak{s}' \mid \mathfrak{s} \cap \mathfrak{s}' \mid \Box\mathfrak{t}$
Interface types	ψ	$::=$	$\sigma \mid \psi \cap \mathfrak{s} \mid \Box\phi \mid \psi \to \psi' \mid \psi \cap \psi'$

Combinatory Logic C2 over C1

$$\mathcal{C} = \{\text{box } X_i : \Box\phi\} \cup \{Y_j : \psi\} \qquad (\Box\phi \text{ by rule of necessitation})$$

Logically, contains modal logic **S4**

$$@ \triangleq \lambda F : \Box(\alpha \to \beta).\lambda X : \Box\alpha.\ \text{letbox } f : \alpha \to \beta = F \text{ in } \text{letbox } x : \alpha = X \text{ in } \text{box } f(x)$$

$$@ : \Box(\alpha \to \beta) \to \Box\alpha \to \Box\beta$$

$$\texttt{eval} \triangleq \lambda X : \Box\alpha.\ \text{letbox } x : \alpha = X \text{ in } x$$

$$\texttt{eval} : \Box\alpha \to \alpha$$

Stratified logics and languages

$$\texttt{Mmap} \triangleq \lambda F : \Box(\alpha \to \beta).\lambda A : \Box\texttt{Array}(\alpha).$$
$$\text{letbox } f : \alpha \to \beta = F \text{ in}$$
$$\text{letbox } a : \texttt{Array}(\alpha) = A \text{ in}$$
$$\text{box}(\texttt{for (i} := 0\texttt{) to sizeof(a)} - 1\{\texttt{a[i]} := \texttt{f(a[i]);}\})$$

$$\texttt{Mmap} : \Box(\alpha \to \beta) \to \Box\texttt{Array}(\alpha) \to \Box\texttt{Array}(\beta)$$

Stratified logics and languages

$$\begin{array}{lcl}
\texttt{Ppar} & \triangleq & \lambda P_1 : \Box\texttt{Proc}.\lambda P_2 : \Box\texttt{Proc}. \\
& & \quad \text{letbox } p_1 : \texttt{Proc} = P_1 \text{ in} \\
& & \quad\quad \text{letbox } p_2 : \texttt{Proc} = P_2 \text{ in} \\
& & \quad\quad\quad \text{box}(p_1 \parallel p_2)
\end{array}$$

$$\texttt{Ppar} \quad : \quad \Box\texttt{Proc} \to \Box\texttt{Proc} \to \Box\texttt{Proc}$$

$$\texttt{Ppar} \quad : \quad \Box(\texttt{Proc} \cap \mathcal{A}_1) \to \Box(\texttt{Proc} \cap \mathcal{A}_2) \to \Box(\texttt{Proc} \cap \mathcal{A}_1 \times \mathcal{A}_2)$$

$$\texttt{Ppar} \quad : \quad (\Box(\texttt{Proc} \cap \mathcal{A}_1) \to \Box(\texttt{Proc} \cap \mathcal{A}_2) \to \Box(\texttt{Proc} \cap \mathcal{A}_1 \times \mathcal{A}_2)) \cap \mathit{ParComp}$$

Example Repository

$$
\Gamma = \{
\begin{array}{lcl}
\texttt{O} & : & \texttt{TrObj} \\
\texttt{Tr} & : & \texttt{TrObj} \to \texttt{D}((\texttt{R},\texttt{R}),\texttt{R},\texttt{R}) \\
\texttt{pos} & : & \texttt{D}((\texttt{R},\texttt{R}),\texttt{R},\texttt{R}) \to ((\texttt{R},\texttt{R}),\texttt{R}) \\
\texttt{cdn} & : & ((\texttt{R},\texttt{R}),\texttt{R}) \to (\texttt{R},\texttt{R}) \\
\texttt{fst} & : & (\texttt{R},\texttt{R}) \to \texttt{R} \\
\texttt{snd} & : & (\texttt{R},\texttt{R}) \to \texttt{R} \\
\texttt{tmp} & : & \texttt{D}((\texttt{R},\texttt{R}),\texttt{R},\texttt{R}) \to \texttt{R} \\
\texttt{cc2pl} & : & ((\texttt{R},\texttt{R}),\texttt{R}) \to ((\texttt{R},\texttt{R}),\texttt{R}) \\
\texttt{cl2fh} & : & \texttt{R} \to \texttt{R}
\end{array}
\}
$$

JR: *Towards Combinatory Logic Synthesis*, BEAT'13 [Rehof, 2013]

Example

$\mathcal{C} = \{$

O $\triangleq$ box `O`	:	$\square\texttt{TrObj}$
Tr $\triangleq$ box `Tr`	:	$\square(\texttt{TrObj} \to \texttt{D}((\texttt{R},\texttt{R}) \cap \mathit{Cart}, \texttt{R} \cap \mathit{Gpst}, \texttt{R} \cap \mathit{Cel}))$
tmp $\triangleq$ box `tmp`	:	$\square(\texttt{D}((\texttt{R},\texttt{R}),\texttt{R},\texttt{R} \cap \mathfrak{a}) \to \texttt{R} \cap \mathfrak{a} \cap \mathit{ms})$

$\}$

$\mathcal{D} = \{$

`@`	:	$\square(\alpha \to \beta) \to \square\alpha \to \square\beta$
$\bullet$	:	$\square(\beta \to \gamma) \to \square(\alpha \to \beta) \to \square(\alpha \to \gamma)$
`cl2fh`	:	$(\square(\texttt{R} \cap \mathit{Cel}) \to \square(\texttt{R} \cap \mathit{Fh})) \cap \mathit{Conv}$
$\diamond$	:	$\square(\alpha \cap \mathit{ms}) \to (\square\alpha \to \square\beta) \cap \mathit{Conv} \to \square(\beta \cap \mathit{ms})$

$\}$

Example

Combinator bindings for $\mathcal{D}$:

$$\bullet \quad \triangleq \quad \lambda G : \Box(\beta \to \gamma).\lambda F : \Box(\alpha \to \beta).\ \text{letbox } f : \alpha \to \beta = F \text{ in } \text{letbox } g : \beta \to \gamma = G \text{ in } \text{box}(\texttt{fn y : } \alpha \texttt{ => (}g\ (f\ \texttt{y}))\texttt{)}$$

$$\texttt{cl2fh} \quad \triangleq \quad \lambda z : \Box\mathtt{R}.\ \text{letbox } u : \mathtt{R} = z \text{ in } \text{box}(\texttt{let x : R = } u \texttt{ in x * (9 div 5) + 32})$$

$$\Diamond \quad \triangleq \quad \lambda z : \Box\alpha.\lambda F : \Box\alpha \to \Box\beta.(F\ z)$$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \Box(\texttt{TrObj} \rightarrow (\texttt{R} \cap \mathit{Cel} \cap \mathit{ms}))$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \Box(\texttt{TrObj} \rightarrow (\texttt{R} \cap \mathit{Cel} \cap \mathit{ms}))$
- Solution: **tmp** • **Tr** : $\Box(\texttt{TrObj} \rightarrow (\texttt{R} \cap \mathit{Cel} \cap \mathit{ms}))$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \square(\texttt{TrObj} \rightarrow (\texttt{R} \cap \mathit{Cel} \cap \mathit{ms}))$
- Solution: **tmp** • **Tr** : $\square(\texttt{TrObj} \rightarrow (\texttt{R} \cap \mathit{Cel} \cap \mathit{ms}))$
- L2-reduction:

$$\textbf{tmp} \bullet \textbf{Tr} \longmapsto^*$$
$$\text{box}(\texttt{fn y : TrObj => (tmp (Tr y))})$$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$
- Solution: $(\mathbf{tmp}\ @\ (\mathbf{Tr}\ @\ \mathbf{O})) \diamond \mathtt{cl2fh}$

Example

- Query: $\mathscr{C} \cup \mathscr{D} \vdash_{C2} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$
- Solution: $(\mathbf{tmp}\ @\ (\mathbf{Tr}\ @\ \mathbf{O})) \diamond \mathtt{cl2fh}$
- L2-reduction:

$$(\mathbf{tmp}\ @\ (\mathbf{Tr}\ @\ \mathbf{O})) \diamond \mathtt{cl2fh} \longmapsto^*$$
$$\mathrm{box}(\mathtt{let\ x : R = tmp\ (Tr\ 0)\ in\ x*(9\ div\ 5) + 32})$$

Example

Let us add to $\mathcal{C}$

$$\textbf{c2f} : \Box((\mathtt{R} \cap \mathit{Cel} \cap \mathit{ms}) \rightarrow (\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms}))$$

- Query: $\mathscr{C}; \mathscr{D} \vdash_{\mathsf{C2}} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$

Example

Let us add to $\mathcal{C}$

$$\textbf{c2f} : \Box((\mathtt{R} \cap \mathit{Cel} \cap \mathit{ms}) \to (\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms}))$$

- Query: $\mathscr{C}; \mathscr{D} \vdash_{\mathsf{C2}} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$
- Solution: $\textbf{c2f} \mathbin{@} ((\textbf{tmp} \bullet \textbf{Tr}) \mathbin{@} \textbf{O})$

Example

Let us add to $\mathcal{C}$

$$\textbf{c2f} : \Box((\mathtt{R} \cap \mathit{Cel} \cap \mathit{ms}) \to (\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms}))$$

- Query: $\mathscr{C}; \mathscr{D} \vdash_{C2} ? : \Box(\mathtt{R} \cap \mathit{Fh} \cap \mathit{ms})$
- Solution: **c2f** @ ((**tmp** • **Tr**) @ **O**)
- After L2-reduction:

$$\text{box}((\texttt{c2f}\ ((\texttt{fn}\ \texttt{y} : \texttt{TrObj} \Rightarrow (\texttt{tmp}\ (\texttt{Tr}\ \texttt{y})))\ \texttt{O}))$$

Outline

Consider the L1-implementation of Quicksort written in a first-order monomorphic Java-like language shown on the following slides. We use the form

$$\texttt{def B foo(A1 x1,..., An xn) \{ ... \}}$$

for locally scoped definitions, as in `def ... { ... } in` e. Such a definition is given the type

$$\texttt{(A1,..., An)} \rightarrow \texttt{B}$$

For function definitions D we assume a metasyntactic L2-operation of the form

$$D.\text{FUN}$$

which extracts the function name defined in expression D,

$$(\texttt{def B foo(A1 x1,..., An xn) \{ ... \}}).\text{FUN} \longmapsto \texttt{foo}$$

and with type $(\texttt{A1,..., An}) \to \texttt{B}$ identical to that of D. We also assume a metalevel operation

$$D.\text{NID}$$

which assigns a globally unique identifier to the definition D and applies the renaming in the scope of D.

L1-implementation of Quicksort (1/2)

```
def int partition(int arr[], int left, int right)
 {
  int i = left, j = right; int tmp;
  int pivot = arr[(left + right) / 2];
  while (i <= j) {
     while (arr[i] < pivot) i++;
     while (arr[j] > pivot) j--;
     if (i <= j) {
        tmp = arr[i]; arr[i] = arr[j];
        arr[j] = tmp; i++; j--;
     }
  };
  return i;
 }
```

L1-implementation of Quicksort (2/2)

```
in
def void QuickSort(int arr[], int left, int right) {
   int index = partition(arr, left, right);
   if (left < index - 1)
      QuickSort(arr, left, index - 1);
   if (index < right)
      QuickSort(arr, index, right);
}
in
def void Sort(int arr[]) {
   Quicksort(arr, 0, sizeof(arr)-1);
}
```

Combinator for Quicksort implemented in L2/L1 (1/3)

$$\mathbf{Q} \quad : \quad \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\epsilon)) \to \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{rev}(\epsilon))) \to \Box((\alpha\,\texttt{[]} \to \texttt{()}) \cap \mathit{SortingFunction})$$

$$\mathbf{Q} \triangleq$$

$$\lambda P : \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\epsilon)).$$
$$\lambda Q : \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{rev}(\epsilon))).$$
$$\text{letbox } p : (\alpha,\alpha) \to \texttt{bool} = P \text{ in}$$
$$\text{letbox } q : (\alpha,\alpha) \to \texttt{bool} = Q \text{ in}$$
$$\text{box}($$
$$p.\text{NID}$$
$$\texttt{in}$$
$$q.\text{NID}$$
$$\texttt{in}$$

Combinator for Quicksort implemented in L2/L1 (2/3)

```
def int partition(α arr[], int left, int right)
 {
  int i = left, j = right; α tmp;
  α pivot = arr[(left + right) / 2];
  while (i <= j) {
     while p.FUN(arr[i], pivot) i++;
     while q.FUN(arr[j], pivot) j--;
     if (i <= j) {
        tmp = arr[i]; arr[i] = arr[j];
        arr[j] = tmp; i++; j--;
     }
  };
  return i;
 }
in
```

Combinator for Quicksort implemented in L2/L1 (3/3)

```
    def void QuickSort(α arr[], int left, int right) {
       int index = partition(arr, left, right);
       if (left < index - 1)
          QuickSort(arr, left, index - 1);
       if (index < right)
          QuickSort(arr, index, right);
    }
    in
    def void Sort(α arr[]) {
       Quicksort(arr, 0, sizeof(arr)-1);
    }
)
```

The combinator **R** implements reversal of a total order. Its semantic type

$$\Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\epsilon)) \to \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{rev}(\epsilon)))$$

expresses this functionality: it maps a total order to a total order, and the result of reversing a relation ϵ satisfies the descriptor $\mathit{rev}(\epsilon)$.

Other L2 combinators in the repository (1/2)

$\mathbf{O}_{\texttt{int}} : \square(((\texttt{int}, \texttt{int}) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{intord}))$

$\mathbf{O}_{\texttt{int}} \triangleq$

```
box(
   def bool intord(int x, int y) { return (x < y); }
)
```

$\mathbf{O}_{\texttt{str}} : \square(((\texttt{string}, \texttt{string}) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{strord}))$

$\mathbf{O}_{\texttt{str}} \triangleq$

```
box(
   def bool strord(string x, string y) {
      return (strlen(x) < strlen(y));
   }
)
```

Other L2 combinators in the repository (2/2)

$$\mathbf{R} \quad : \quad \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\epsilon)) \to \Box(((\alpha,\alpha) \to \texttt{bool}) \cap \mathit{TotalOrder} \cap \mathit{id}(\mathit{rev}(\epsilon)))$$

$$\mathbf{R} \triangleq$$

$\lambda P : \Box((\alpha,\alpha) \to \texttt{bool})$.
letbox $p : (\alpha,\alpha) \to \texttt{bool} = P$ in
box(
p.NID
in
def bool revord(α x, α y) { return p.FUN(y,x); }
)

We can now synthesize from the combined repository by collecting the type declarations of the combinator symbols shown on the previous slides into a combinatory type environment $\mathscr{C}$, that is

$$\mathscr{C} = \{\mathbf{Q} : \tau_{\mathbf{Q}}, \mathbf{O}_{\mathtt{int}} : \tau_{\mathbf{O}_{\mathtt{int}}}, \mathbf{O}_{\mathtt{str}} : \tau_{\mathbf{O}_{\mathtt{str}}}, \mathbf{R} : \tau_{\mathbf{R}}\}$$

To synthesize a sorting program on arrays of strings we execute the combinatory logic synthesizer on the input inhabitation goal:

$$\mathscr{C} \vdash ? : \square((\mathtt{string[]} \to ()) \cap \mathit{SortingFunction})$$

$$\mathscr{C} \vdash \mathbf{Q}\ \mathbf{O}_{\mathtt{str}}(\mathbf{R}\ \mathbf{O}_{\mathtt{str}}) : \square((\mathtt{string[]} \to ()) \cap \mathit{SortingFunction})$$

Normalizing the inhabitant we get

$$\mathbf{Q}\ \mathbf{O}_{\mathtt{str}}(\mathbf{R}\ \mathbf{O}_{\mathtt{str}}) \longmapsto^* \mathrm{box}\ N$$

for a well-typed L1-expression N, in accordance with the property of Eliminability. Next slide shows the term box N resulting from the reduction.

Quicksort implementation synthesized from repository (1/3)


```
box(
   def bool strord1(string x, string y) {
      return (strlen(x) < strlen(y));
   }
   in
   def bool strord2(string x, string y) {
      return (strlen(x) < strlen(y));
   }
   in
```

Quicksort implementation synthesized from repository (2/3)


```
def bool revord1(string x, string y) { return strord2(y,x)
in
def int partition(string arr[], int left, int right)
    {
     int i = left, j = right; string tmp;
     string pivot = arr[(left + right) / 2];
     while (i <= j) {
        while strord1(arr[i], pivot) i++;
        while revord1(arr[j], pivot) j--;
        if (i <= j) {
           tmp = arr[i]; arr[i] = arr[j];
           arr[j] = tmp; i++; j--;
        }
     };
```

Quicksort implementation synthesized from repository (3/3)


```
    in
    def void QuickSort(string arr[], int left, int right) {
       int index = partition(arr, left, right);
       if (left < index - 1)
          QuickSort(arr, left, index - 1);
       if (index < right)
          QuickSort(arr, index, right);
    }
    in
    def void Sort(string arr[]) {
       Quicksort(arr, 0, sizeof(arr)-1);
    }
)
```

Demonstration

Outline

Some Main Points

- A new approach to component-oriented synthesis
- An “Aristotelian” rather than “Fregean” approach
- Types as logic programs that compute compositions
- Easily applicable to native API’s
- Very large set of application scenarios
- Robust foundations in proof theory
- Implemented in **(CL)S**-framework
- Applied in several experiments

Ongoing & Future Work

- Application to connector synthesis
- Application to OO-synthesis (traits, dependency injection)
- Combinatory process synthesis
- Algorithm engineering
- Constrained types $Q(\vec{\alpha}) \Rightarrow \phi$
- Automatic software configuration
- Stratified logics (multimodal logic)
- Component-oriented synthesis in theorem-proving?
- ...


Davies, R. and Pfenning, F. (2001).
A Modal Analysis of Staged Computation.
Journal of the ACM, 48(3):555–604.


Düdder, B., Martens, M., and Rehof, J. (2014).
Staged Composition Synthesis.
In *Proceedings of ESOP'14*, volume 8410 of *LNCS*, pages 67–86. Springer.


Rehof, J. (2013).
Towards Combinatory Logic Synthesis.
In *BEAT'13, 1st International Workshop on Behavioural Types*. ACM.