GRADUAL SET-THEORETIC TYPES

HOW TO ADD SET-THEORETIC TYPES TO A GRADUAL TYPE SYSTEM

Victor **Lanvin**
Supervised by Giuseppe **Castagna**
IRIF

MOTIVATIONS

SEMANTICS

COMPILATION

MOTIVATIONS INTERSECTION TYPES

```
let plus_int : int -> int -> int = fun x y -> x + y;;

let plus_float : float -> float -> float = fun x y -> x +. y;;

let plus :                                              = function
  | x : int -> plus_int x
  | x : float -> plus_float x;;
```

MOTIVATIONS INTERSECTION TYPES

```
let plus_int : int -> int -> int = fun x y -> x + y;;

let plus_float : float -> float -> float = fun x y -> x +. y;;

let plus :                    ???                    = function
  | x : int -> plus_int x
  | x : float -> plus_float x;;
```

MOTIVATIONS INTERSECTION TYPES

```
let plus_int : int -> int -> int = fun x y -> x + y;;

let plus_float : float -> float -> float = fun x y -> x +. y;;

let plus : (int -> int -> int) && (float -> float -> float) = function
  | x : int -> plus_int x
  | x : float -> plus_float x;;
```

MOTIVATIONS UNION TYPES

```
let print_int : int -> unit = ...;;

let print_float : float -> unit = ...;;

let print_string : string -> unit = ...;;

let print :                                  = function
  | x : int -> print_int x
  | x : float -> print_float x
  | x : string -> print_string x
```

MOTIVATIONS UNION TYPES

```
let print_int : int -> unit = ...;;

let print_float : float -> unit = ...;;

let print_string : string -> unit = ...;;

let print :              ???              = function
  | x : int -> print_int x
  | x : float -> print_float x
  | x : string -> print_string x
```

MOTIVATIONS UNION TYPES

```
let print_int : int -> unit = ...;;

let print_float : float -> unit = ...;;

let print_string : string -> unit = ...;;

let print : (int V float V string) -> unit = function
  | x : int -> print_int x
  | x : float -> print_float x
  | x : string -> print_string x
```

MOTIVATIONS GRADUAL TYPES (1)

```
let rec map :
  ('a :: 'a list -> ('a -> 'b) -> 'b :: 'b list)
  && ([] -> ('a -> 'b) -> []) = function
  | h::t -> (fun (('a -> 'b) -> 'b list) f -> (f h) :: (map t f))
  | []   -> (fun (('a-> 'b) -> []) _ -> []);;
```

MOTIVATIONS GRADUAL TYPES (1)

```
let rec map :
  ('a :: 'a list -> ('a -> 'b) -> 'b :: 'b list)
  && ([] -> ('a -> 'b) -> []) = function
  | h::t -> (fun (('a -> 'b) -> 'b list) f -> (f h) :: (map t f))
  | []   -> (fun (('a-> 'b) -> []) _ -> []);;
```

MOTIVATIONS GRADUAL TYPES (1)

```
let rec map :
  ('a :: 'a list -> ('a -> 'b) -> 'b :: 'b list)
  && ([] -> ('a -> 'b) -> []) = function
  | h::t -> (fun (('a -> 'b) -> 'b list) f -> (f h) :: (map t f))
  | []   -> (fun (('a-> 'b) -> []) _ -> []);;
```

```
let rec map (l : ? list) (f : ? -> ?) : ? list =
  match l with
  | [] -> []
  | h::t -> (f h) :: (map t f);;
```

MOTIVATIONS GRADUAL TYPES (1)

```
let rec map :
  ('a :: 'a list -> ('a -> 'b) -> 'b :: 'b list)
  && ([] -> ('a -> 'b) -> []) = function
  | h::t -> (fun (('a -> 'b) -> 'b list) f -> (f h) :: (map t f))
  | []   -> (fun (('a-> 'b) -> []) _ -> []);;
```

▶ SAFE

```
let rec map (l : ? list) (f : ? -> ?) : ? list =
  match l with
  | [] -> []
  | h::t -> (f h) :: (map t f);;
```

▶ FAST

MOTIVATIONS GRADUAL TYPES (2)

```
let f : Int -> Int = ...
let g : Bool -> Bool = ...
let condition = ref false

let func (x : ?) : ? =
  if !condition then
    f x
  else
    g x
```

MOTIVATIONS GRADUAL TYPES (2)

```
let f : Int -> Int = ...
let g : Bool -> Bool = ...
let condition = ref false

let func (x : ?) : ? =
  if !condition then
    f x
  else
    g x
```

WHICH TYPES ?

MOTIVATIONS GRADUAL TYPES (2)

```
let f : Int -> Int = ...
let g : Bool -> Bool = ...
let condition = ref false

let func (x : ?) : ? =
  if !condition then
    f x
  else
    g x
```

WHICH TYPES ?

- THE COMPILER AUTOMATICALLY ADDS THE NECESSARY TYPE TESTS.

MOTIVATIONS THE TYPE SYSTEM [Siek et al. '06]

▶ STATIC TYPES

$$t \in \mathrm{SType} ::= t \vee t \mid t \wedge t \mid \neg t \mid t \to t \mid \mathit{Int} \mid \mathit{Bool} \mid \mathbb{0} \mid \mathbb{1}$$

MOTIVATIONS THE TYPE SYSTEM [Siek et al. '06]

▶ STATIC TYPES

$t \in \mathrm{SType} ::= t \vee t \mid t \wedge t \mid \neg t \mid t \to t \mid \mathit{Int} \mid \mathit{Bool} \mid \mathbb{0} \mid \mathbb{1}$

▶ GRADUAL TYPES

$\tau \in \mathrm{GType} ::= ? \mid \tau \vee \tau \mid \tau \wedge \tau \mid \neg \tau \mid \tau \to \tau \mid \mathit{Int} \mid \mathit{Bool} \mid \mathbb{0} \mid \mathbb{1}$

MOTIVATIONS THE TYPE SYSTEM [Siek et al. '06]

▶ STATIC TYPES

$$t \in \mathrm{SType} ::= t \vee t \mid t \wedge t \mid \neg t \mid t \to t \mid \mathit{Int} \mid \mathit{Bool} \mid \mathbb{0} \mid \mathbb{1}$$

▶ GRADUAL TYPES

$$\tau \in \mathrm{GType} ::= ? \mid \tau \vee \tau \mid \tau \wedge \tau \mid \neg \tau \mid \tau \to \tau \mid \mathit{Int} \mid \mathit{Bool} \mid \mathbb{0} \mid \mathbb{1}$$

▶ GRADUALLY-TYPED LAMBDA CALCULUS

$$e \in \mathrm{Term} ::= \lambda^{\{\sigma_i \to \tau_i \mid i \in I\}} x.e \mid e\ e \mid (e \in t)?e : e \mid x \mid c$$

MOTIVATIONS

SEMANTICS

COMPILATION

SEMANTICS CONCRETISATION [Garcia et al. POPL '16]

- **POSSIBLE INTERPRETATIONS :**

$$\gamma : \mathrm{GType} \to \mathcal{P}(\mathrm{SType})$$

SEMANTICS CONCRETISATION [Garcia et al. POPL '16]

- **POSSIBLE INTERPRETATIONS :**

$$\gamma : \mathrm{GType} \to \mathcal{P}(\mathrm{SType})$$

$$\gamma(?) = \mathrm{SType}$$

SEMANTICS CONCRETISATION [Garcia et al. POPL '16]

- **POSSIBLE INTERPRETATIONS :**

$$\gamma : \mathrm{GType} \to \mathcal{P}(\mathrm{SType})$$

$$\gamma(?) = \mathrm{SType}$$

- **EXAMPLE :**

$$\gamma(? \to Int) = \{T \to Int \mid T \in \mathrm{SType}\}$$

SEMANTICS ABSTRACTION AND INTERPRETATION

- **ABSTRACTION :** $\alpha : \mathcal{P}(\mathrm{SType}) \to \mathrm{GType}$

SEMANTICS ABSTRACTION AND INTERPRETATION

- **ABSTRACTION :** $\alpha : \mathcal{P}(\mathrm{SType}) \to \mathrm{GType}$

- **PREDICATES :** $\mathcal{P}(\mathrm{SType}) \xleftarrow{\gamma} \mathrm{GType}$

  $\exists P \qquad\qquad P_G$

SEMANTICS ABSTRACTION AND INTERPRETATION

- **ABSTRACTION :** $\alpha : \mathcal{P}(\mathrm{SType}) \to \mathrm{GType}$

- **PREDICATES :**
$$\begin{array}{ccc} \mathcal{P}(\mathrm{SType}) & \xleftarrow{\ \gamma\ } & \mathrm{GType} \\ \exists P & & P_G \end{array}$$

- **FUNCTIONS :**
$$\begin{array}{ccc} \mathcal{P}(\mathrm{SType}) & \xleftarrow{\ \gamma\ } & \mathrm{GType} \\ f \downarrow & & \downarrow f_G \\ \mathcal{P}(\mathrm{SType}) & \xrightarrow[\ \alpha\ ]{} & \mathrm{GType} \end{array}$$

SEMANTICS SUBTYPING

$$\sigma \leq_G \tau \stackrel{def}{\Longleftrightarrow} \exists (s,t) \in \gamma(\sigma) \times \gamma(\tau), s \leq t$$

SEMANTICS SUBTYPING

$$\sigma \leq_G \tau \stackrel{def}{\Longleftrightarrow} \exists (s,t) \in \gamma(\sigma) \times \gamma(\tau), s \leq t$$

EXAMPLES :

$$? \leq_G Int \qquad\qquad Int \to Int \leq_G Nat \to ?$$

SEMANTICS SUBTYPING

$$\sigma \leq_G \tau \stackrel{def}{\Longleftrightarrow} \exists (s,t) \in \gamma(\sigma) \times \gamma(\tau), s \leq t$$

EXAMPLES :

$$? \leq_G Int \qquad\qquad Int \to Int \leq_G Nat \to ?$$

THEOREM :

Deciding gradual subtyping is equivalent to deciding static subtyping, without any increase in complexity.

SEMANTICS SUBTYPING

$$\sigma \leq_G \tau \stackrel{def}{\Longleftrightarrow} \exists (s,t) \in \gamma(\sigma) \times \gamma(\tau), s \leq t$$

EXAMPLES :

$$? \leq_G Int \qquad\qquad Int \to Int \leq_G Nat \to ?$$

THEOREM :

Deciding gradual subtyping is equivalent to deciding static subtyping, without any increase in complexity.

$$\sigma \leq_G \tau \iff \sigma^{\Downarrow} \leq \tau^{\Uparrow}$$

SEMANTICS FUNCTIONS (1) [Frisch et al. JACM '08]

$$t = (Int \to Int)$$

SEMANTICS FUNCTIONS (1) [Frisch et al. JACM '08]

$$t = (Int \rightarrow Int)$$

DOMAIN RESULT

SEMANTICS FUNCTIONS (1) [Frisch et al. JACM '08]

$$t = (Int \to Int) \wedge (Bool \to Bool)$$

SEMANTICS FUNCTIONS (1) [Frisch et al. JACM '08]

$$t = (Int \to Int) \wedge (Bool \to Bool)$$

$$\simeq (Int \vee Bool) \to (Int \vee Bool) \ \textbf{?}$$

SEMANTICS FUNCTIONS (1) [Frisch et al. JACM '08]

$$t = (Int \to Int) \wedge (Bool \to Bool)$$

$$\not\wedge$$

$$\simeq \cancel{(Int \vee Bool) \to (Int \vee Bool)}$$

$$dom(t) = Int \vee Bool$$

$$t@Int = Int \qquad t@Bool = Bool$$

$$t@(Int \vee Bool) = Int \vee Bool$$

SEMANTICS PROBLEM

$(? \to Bool) \wedge (Int \to Int)$ **DOMAIN ?**

SEMANTICS PROBLEM

$$(? \to Bool) \wedge (Int \to Int) \quad \textbf{DOMAIN ?}$$

$$\downarrow \gamma$$

$$\{(T \to Bool) \wedge (Int \to Int) \mid T \in \mathrm{SType}\}$$

SEMANTICS PROBLEM

$$(? \to Bool) \wedge (Int \to Int) \quad \textbf{DOMAIN ?}$$

$$\downarrow \gamma$$

$$\{(T \to Bool) \wedge (Int \to Int) \mid T \in \mathrm{SType}\}$$

$$\downarrow dom$$

$$\{T \vee Int \mid T \in \mathrm{SType}\}$$

SEMANTICS PROBLEM

$$(? \to Bool) \wedge (Int \to Int) \quad \textbf{DOMAIN ?}$$

$$\downarrow \gamma$$

$$\{(T \to Bool) \wedge (Int \to Int) \mid T \in \mathrm{SType}\}$$

$$\downarrow dom$$

$$\{T \vee Int \mid T \in \mathrm{SType}\}$$

$$\downarrow \alpha$$

$$? \vee Int$$

SEMANTICS PROBLEM

$$(? \to Bool) \wedge (Int \to Int)$$ **DOMAIN ?**

$$\downarrow \gamma$$

$$\{(T \to Bool) \wedge (Int \to Int) \mid T \in \mathrm{SType}\}$$

$$\downarrow dom$$

$$\{T \vee Int \mid T \in \mathrm{SType}\}$$

$$\downarrow \alpha$$ **DEFINITION ?**

$$? \vee Int$$

SEMANTICS FUNCTIONS (2)

SOLUTION : FINITE CONCRETISATION...

$$\mathcal{A} : \mathrm{GType}_{\underline{\leq}\mathbb{0}\to\mathbb{1}} \to \mathcal{P}_f(\mathrm{GType})$$

SEMANTICS FUNCTIONS (2)

SOLUTION : FINITE CONCRETISATION...

$$\mathcal{A} : \mathrm{GType}_{\underline{\leq}\mathbb{0}\to\mathbb{1}} \to \mathcal{P}_f(\mathrm{GType})$$

... AND FINITE ABSTRACTION :

$$\alpha_f : \mathcal{P}_f(\mathrm{GType}) \to \mathrm{GType}$$

$$\alpha_f(S) = \bigvee_{\sigma \in S} \sigma$$

SEMANTICS FUNCTIONS (2)

SOLUTION : FINITE CONCRETISATION...

$$\mathcal{A} : \mathrm{GType}_{\underline{\leq}\mathbb{0}\to\mathbb{1}} \to \mathcal{P}_f(\mathrm{GType})$$

... AND FINITE ABSTRACTION :

$$\alpha_f : \mathcal{P}_f(\mathrm{GType}) \to \mathrm{GType}$$

$$\alpha_f(S) = \bigvee_{\sigma \in S} \sigma$$

- $\mathcal{A}$ DOES SOME APPROXIMATIONS

SEMANTICS EXAMPLE

$$(? \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int$$

SEMANTICS EXAMPLE

$$(? \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int$$

$$\downarrow \mathcal{A}$$

$$\{(\mathbb{0} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int,$$

$$(\mathbb{1} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int\}$$

SEMANTICS EXAMPLE

$$(? \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int$$

$$\downarrow \mathcal{A}$$

$$\{(\mathbb{0} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int,$$

$$(\mathbb{1} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int\}$$

$$\downarrow dom$$

$$\{Int, Int \wedge Bool\}$$

SEMANTICS EXAMPLE

$$(? \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int$$

$$\downarrow \mathcal{A}$$

$$\{(\mathbb{0} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int,$$

$$(\mathbb{1} \to Bool) \wedge (Int \to Int) \;\; @ \;\; Int\}$$

$$\downarrow dom$$

$$\{Int, Int \wedge Bool\}$$

$$\downarrow \alpha_f$$

$$Int \vee (Int \wedge Bool) \simeq Int$$

SEMANTICS TYPING EXAMPLES

$$\frac{\forall(\sigma \to \tau) \in \mathbb{I}, \quad \Gamma, x : \sigma \vdash e : \tau}{\lambda^{\mathbb{I}} x.\ e : \bigwedge_{(\sigma\to\tau)\in\mathbb{I}} \sigma \to \tau}\ (\mathsf{TAbs})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad {\tau_1}^{\Downarrow} \leq \mathbb{0} \to \mathbb{1} \quad {\tau_2}^{\Downarrow} \leq gdom(\tau_1)}{\Gamma \vdash e_1\ e_2 : \tau_1 @ \tau_2}\ (\mathsf{TApp})$$

$$\frac{\Gamma \vdash e : \tau \quad \begin{cases} \tau^{\Uparrow} \not\leq \neg t & \Longrightarrow\ \Gamma \vdash e_1 : \sigma_1 \\ \tau^{\Uparrow} \not\leq t & \Longrightarrow\ \Gamma \vdash e_2 : \sigma_2 \end{cases}}{\Gamma \vdash ((e \in t)?e_1 : e_2) : \sigma_1 \vee \sigma_2}\ (\mathsf{TCase})$$

MOTIVATIONS

SEMANTICS

COMPILATION

COMPILATION CAST LANGUAGE

- **SYNTAX :**

$$e ::= \lambda^{\{\sigma_i \to \tau_i | i \in I\}}_{\langle \tau \rangle} x.e \mid e\ e \mid (e \in t)?e : e \mid x \mid c \mid \langle \tau \rangle e$$

COMPILATION CAST LANGUAGE

▶ SYNTAX :

$$e ::= \lambda^{\{\sigma_i \to \tau_i | i \in I\}}_{\langle \tau \rangle} x.e \mid e\ e \mid (e \in t)?e : e \mid x \mid c \mid \langle \tau \rangle e$$

$$v ::= \lambda^{\{\sigma_i \to \tau_i | i \in I\}}_{\langle \tau \rangle} x.e \mid c$$

$$\mathcal{E} ::= CastError$$

COMPILATION CAST LANGUAGE

▶ SYNTAX :

$$e ::= \lambda^{\{\sigma_i \to \tau_i | i \in I\}}_{\langle \tau \rangle} x.e \mid e\ e \mid (e \in t)?e : e \mid x \mid c \mid \langle \tau \rangle e$$

$$v ::= \lambda^{\{\sigma_i \to \tau_i | i \in I\}}_{\langle \tau \rangle} x.e \mid c$$

$$\mathcal{E} ::= CastError$$

▶ EXAMPLES :

$$\langle Int \rangle (true \ \&\& \ false) \mapsto \langle Int \rangle false \mapsto CastError$$

$$\langle ? \rangle 3 \mapsto 3$$

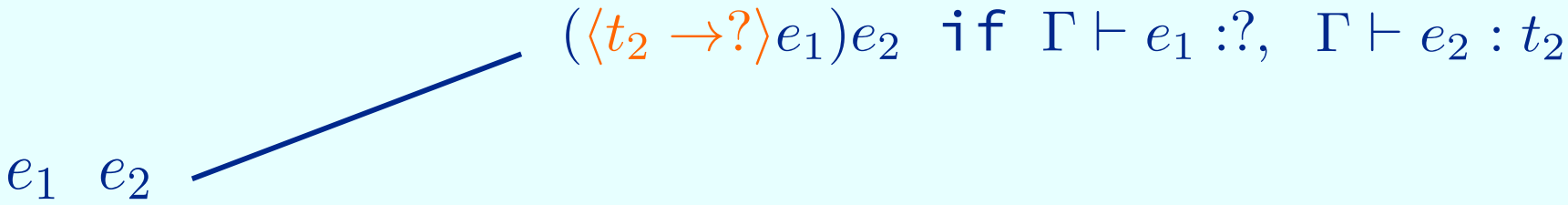
COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?

$e_1 \quad e_2$

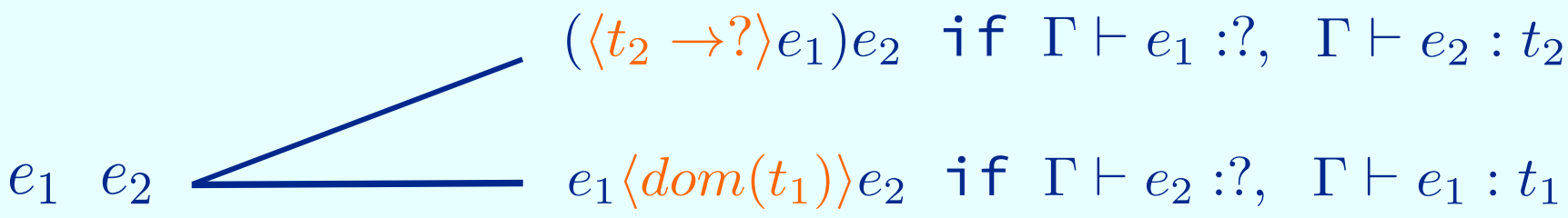
COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?



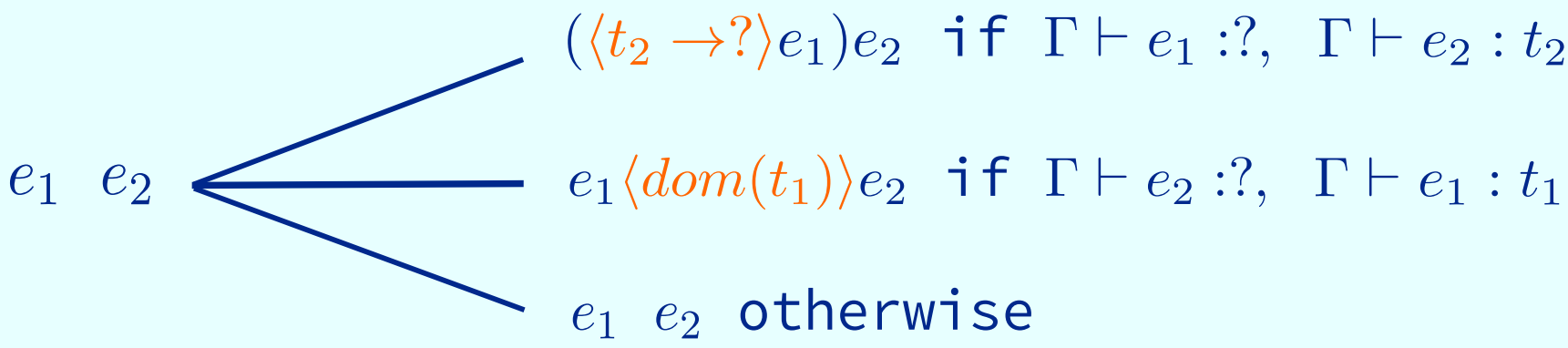
COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?



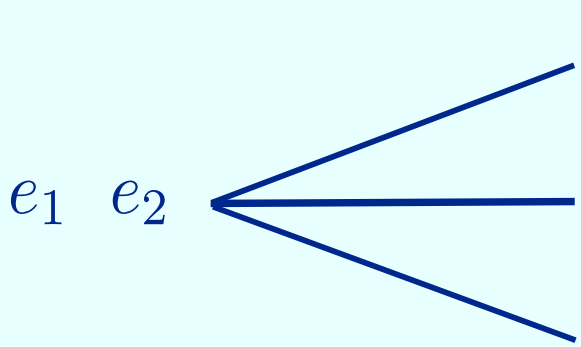
COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?



COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?



$$(\langle t_2 \to ? \rangle e_1)e_2 \quad \texttt{if} \quad \Gamma \vdash e_1 : ?, \quad \Gamma \vdash e_2 : t_2$$

$$e_1 \langle dom(t_1) \rangle e_2 \quad \texttt{if} \quad \Gamma \vdash e_2 : ?, \quad \Gamma \vdash e_1 : t_1$$

$$e_1 \ e_2 \ \texttt{otherwise}$$

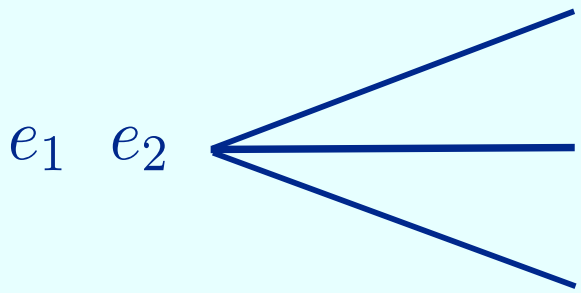
AND WITH SET-THEORETIC TYPES ?

$$\Gamma \vdash e_1 : (Int \to Int) \wedge ? \qquad \Gamma \vdash e_2 : Bool$$

$$\Gamma \vdash e_2 : Int$$

COMPILATION CAST INSERTION (1)

HOW TO COMPILE APPLICATIONS ?



$$e_1\ e_2$$

$$(\langle t_2 \to ? \rangle e_1) e_2 \quad \texttt{if} \quad \Gamma \vdash e_1 : ?,\ \ \Gamma \vdash e_2 : t_2$$

$$e_1 \langle dom(t_1) \rangle e_2 \quad \texttt{if} \quad \Gamma \vdash e_2 : ?,\ \ \Gamma \vdash e_1 : t_1$$

$$e_1\ e_2 \quad \texttt{otherwise}$$

AND WITH SET-THEORETIC TYPES ?

$$\Gamma \vdash e_1 : (Int \to Int) \wedge ? \qquad \Gamma \vdash e_2 : Bool \quad \textbf{CAST}$$

$$\Gamma \vdash e_2 : Int \quad \textbf{NO CAST}$$

COMPILATION CAST INSERTION (2)

▶ SAFE DOMAIN, POSSIBLE DOMAIN :

$$dom^C : \mathrm{GType} \to \mathrm{SType} \qquad dom^P : \mathrm{GType} \to \mathrm{SType}$$

$$dom^C(?) = \mathbb{0} \qquad dom^P(?) = \mathbb{1}$$

COMPILATION CAST INSERTION (2)

▶ SAFE DOMAIN, POSSIBLE DOMAIN :

$$dom^C : \mathrm{GType} \to \mathrm{SType} \qquad dom^P : \mathrm{GType} \to \mathrm{SType}$$

$$dom^C(?) = \mathbb{0} \qquad dom^P(?) = \mathbb{1}$$

▶ TWO CRITERIA :

Argument always in the safe domain

Argument always in the possible domain

COMPILATION CAST INSERTION (2)

▶ SAFE DOMAIN, POSSIBLE DOMAIN :

$$dom^C : \mathrm{GType} \to \mathrm{SType} \qquad dom^P : \mathrm{GType} \to \mathrm{SType}$$

$$dom^C(?) = \mathbb{0} \qquad dom^P(?) = \mathbb{1}$$

▶ TWO CRITERIA :

Argument always in the safe domain	NO CAST
Argument always in the possible domain	CAST THE FUNCTION

COMPILATION CAST INSERTION (2)

▶ SAFE DOMAIN, POSSIBLE DOMAIN :

$$dom^C : \mathrm{GType} \to \mathrm{SType} \qquad dom^P : \mathrm{GType} \to \mathrm{SType}$$

$$dom^C(?) = \mathbb{0} \qquad dom^P(?) = \mathbb{1}$$

▶ TWO CRITERIA :

Argument always in the safe domain — NO CAST

Argument always in the possible domain — CAST THE FUNCTION

OTHERWISE, CAST THE PARAMETER

COMPILATION LAZY CASTS

- **WITHOUT LAZY EVALUATION :**

$$\langle \sigma \to \tau \rangle v \mapsto \lambda z : \sigma . \langle \tau \rangle (v(\langle dom(\tau') \rangle) z) \qquad \text{où} \quad \emptyset \vdash v : \tau'$$

  - **INEFFICIENT [Takikawa et al. POPL '16]**

COMPILATION LAZY CASTS

WITHOUT LAZY EVALUATION :

$$\langle \sigma \to \tau \rangle v \mapsto \lambda z : \sigma . \langle \tau \rangle (v(\langle dom(\tau') \rangle) z) \qquad \text{où} \quad \emptyset \vdash v : \tau'$$

▶ INEFFICIENT [Takikawa et al. POPL '16]

WITH LAZY EVALUATION :

$$\langle \sigma \to \tau \rangle \lambda^{\mathbb{I}}_{\langle \tau' \rangle} x.\ \ e \mapsto \lambda^{\mathbb{I}}_{\langle \sigma \to \tau \rangle} x.\ \ e$$

$$(\lambda^{\mathbb{I}}_{\langle \sigma \to \tau \rangle} x.\ \ e) v \mapsto \langle \tau \rangle (e[x := \langle dom(\mathbb{I}) \rangle v])$$

▶ EFFICIENT?

COMPILATION RESULTS (1)

SOUNDNESS OF THE CAST LANGUAGE :

Every convergent well-typed term of the cast language reduces either to a value or to a cast error.

COMPILATION RESULTS (1)

SOUNDNESS OF THE CAST LANGUAGE :

Every convergent well-typed term of the cast language reduces either to a value or to a cast error.

STATIC SOUNDNESS :

Every convergent well-typed term of the cast language that does not contain gradual types nor casts reduces to a value.

COMPILATION RESULTS (2)

SOUNDNESS OF THE COMPILATION :

Every well-typed term of the GTLC compiles to a well-typed term (of the same type) of the cast language.

COMPILATION RESULTS (2)

SOUNDNESS OF THE COMPILATION :

Every well-typed term of the GTLC compiles to a well-typed term (of the same type) of the cast language.

SOUNDNESS OF THE GTLC :

Every well-typed convergent term of the GTLC reduces either to a value or to a cast error.

COMPILATION RESULTS (2)

SOUNDNESS OF THE COMPILATION :

Every well-typed term of the GTLC compiles to a well-typed term (of the same type) of the cast language.

SOUNDNESS OF THE GTLC :

Every well-typed convergent term of the GTLC reduces either to a value or to a cast error.

STATIC SOUNDNESS OF THE GTLC :

Every well-typed convergent term of the GTLC that does not contain any gradual type reduces to a value.

CONCLUSION RESULTS AND FUTURE WORK

+ EFFICIENT SYSTEM
+ STATICALLY AND GRADUALLY SOUND
+ FEW APPROXIMATIONS AND RESTRICTIONS

- VERY TAILORED APPROACH

CONCLUSION RESULTS AND FUTURE WORK

+ EFFICIENT SYSTEM

+ STATICALLY AND GRADUALLY SOUND

+ FEW APPROXIMATIONS AND RESTRICTIONS

- VERY TAILORED APPROACH

► IMPLEMENT THE LANGUAGE

► BENCHMARKS

► GENERALIZE THE APPROACH

CONCLUSION BIBLIOGRAPHY

[1] Siek, J. G., & Taha, W. (2006, September). Gradual typing for functional languages. In *Scheme and Functional Programming Workshop* (Vol. 6, pp. 81-92).

[2] Frisch, A., Castagna, G., & Benzaken, V. (2002). Semantic subtyping. In *Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on* (pp. 137-146). IEEE.

[3] Garcia, R., Clark, A. M., & Tanter, É. (2016). Abstracting gradual typing. *ACM SIGPLAN Notices*, 51(1), 429-442.