A Calculus with Recursive Types, Record Concatenation and Subtyping

Yaoda Zhou (The University of Hong Kong)

Bruno C. d. S. Oliveira (The University of Hong Kong)

Andong Fan (Zhejiang University)

Record calculi

- Record calculi with a concatenation operator
- give the semantics of object-oriented languages with multiple inheritance
- E.g. the semantics of the Obliq language (Cook and Palsberg 1989, Cardelli 1995)

Record calculi

- Record calculi with a concatenation operator
- give the semantics of object-oriented languages with multiple inheritance
- E.g. the semantics of the Obliq language (Cook and Palsberg1989, Cardelli 1995)
- Record calculi with a concatenation operator and subtyping?
- Subtyping can hide static type information that is needed to correctly model record concatenation (Cardelli and Mitchell 1991)

An example

- What should the program evaluate to because of the extra field ***y*** in the first argument?

```
let f2 (r:{x:Int}) (s:{y:Bool}) : {x:Int} & {y:Bool}
  = r,,s in f2 ({x=3, y=4}) ({y=true, x=false})
```

An example

- What should the program evaluate to because of the extra field ***y*** in the first argument?

```
let f2 (r:{x:Int}) (s:{y:Bool}) : {x:Int} & {y:Bool}
  = r,,s in f2 ({x=3, y=4}) ({y=true, x=false})
```

- Option 1:
  - Symmetric record concatenation that only allows the concatenation of records without conflicts
  - The example cannot pass typed check
- Option 2:
  - Asymmetric record concatenation that employs left or right bias concatenation
  - The example will evaluate to ***{x=3, y=4}***

“we should now feel compelled to define R & S only when R and S are disjoint: that is when any field present in an element of R is absent from every element of S, and vice versa.”

— Cardelli and Mitchell 1991

Disjoint intersection types

- Calculi with disjoint intersection types (Oliveira et al. 2016) and a merge operator (Dunfield 2014) offers a solution to the Cardelli and Mitchell's problem for concatenation
- The λ_i calculus (Oliveira et al. 2016, Bi et al. 2018) adopts disjointness and restricts subsumption to address the challenges of symmetric concatenation/merge
- An important limitation of existing calculi: lack of recursive types

Adding iso-recursive types to λ_i

- Object interface:

$$Exp := \mu Exp\,.\,\{eval : Int,\ dbl : Exp,\ eq : Exp \rightarrow bool\}$$

- Auxiliary functions:

$$eval'\ e = (\text{unfold}\ [Exp]\ e)\,.\,eval \qquad dbl'\ e = (\text{unfold}\ [Exp]\ e)\,.\,dbl$$

$$eq'\ e_1\ e_2 = (\text{unfold}\ [Exp]\ e_1)\,.\,eq\ e_2$$

- Recursive functions to encode expressions:

$$lit\ n = \text{fold}\ [Exp]\ \{\ eval := n,\ dbl := lit(n * 2),\ eq := \lambda e'\,.\,(eval'\ e' = n)\ \}$$

$$add\ e_1\ e_2 = \text{fold}\ [Exp]\ \{\ eval := eval'\ e_1 + eval'\ e_2,\ dbl := add\ (dbl'\ e_1)\ (dbl'\ e_2),$$
$$eq := \lambda e'\,.\,(eval'\ e' = eval'\ e_1 + eval'\ e_2)\ \}$$

Adding iso-recursive types to λ_i

- Recursive functions to encode expressions:

$$lit\ n = \text{fold}\ [Exp]\ \{\ eval := n,\ dbl := lit(n * 2),\ eq := \lambda e'.(eval'\ e' = n)\ \}$$

$$add\ e_1\ e_2 = \text{fold}\ [Exp]\ \{\ eval := eval'\ e_1 + eval'\ e_2,\ dbl := add\ (dbl'\ e_1)\ (dbl'\ e_2), \\ eq := \lambda e'.(eval'\ e' = eval'\ e_1 + eval'\ e_2)\ \}$$

- Check if $2 * 7 = 2 * (3 + 4)$:

$$\left.\begin{array}{l} e_1 := lit\ 7 \\ e_2 := add\ (lit\ 3)\ (lit\ 4) \end{array}\right\}\quad eq'\ (dbl'\ e_1)\ (dbl'\ e_2)$$

The λ_i^μ calculus

- In this work, we studies a calculus λ_i^μ which combines:
  - Iso-recursive types
  - (Disjoint) intersection types with a merge operator
  - Top-like types
  - Bottom types
- We prove that subtyping relation is
  - Transitive
  - Decidable

Iso-recursive subtyping

- For checking if two iso-recursive types are subtype, we choose the recent developed nominal unfolding rules (Zhou et al. 2022):

$$\frac{\Gamma, \alpha \vdash [\alpha \mapsto \{\alpha' : \ \tau\}]\ \tau \leq [\alpha \mapsto \{\alpha' : \ \sigma\}]\ \sigma \qquad \text{fresh } a'}{\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\alpha\,.\,\sigma}$$

- The labelled types $\{\alpha : \tau\}$ have two usage in this work:
  - The label provides a unique identifier for recursive types being compared.
  - The record types and records can be encoded as the combination of labelled types, intersection types and merge (Dunfield 2014).

Top-like types and disjointness

- A type A is top-like type if it is the supertype of ⊤

$$\frac{}{\rceil \top \lceil} \qquad \frac{\rceil \tau \lceil}{\rceil \tau \to \sigma \lceil} \qquad \frac{\rceil \tau \lceil \quad \rceil \sigma \lceil}{\rceil \tau \,\&\, \sigma \lceil} \qquad \frac{\rceil \tau \lceil}{\rceil \mu\alpha \,.\, \tau \lceil} \qquad \frac{\rceil \tau \lceil}{\rceil \{\alpha : \tau\} \lceil}$$

- Specification of disjointness:

  Two types τ and σ are disjoint

  if all common supertypes of them are top-like types

- Allowing a larger set of top-like types enables more types to be disjoint

Disjointness

- Without top-like types, any two function types are disjoint:

  $\bot \to \top$ is the supertype of all function types

- Without top-like types, any two recursive types are disjoint

  $\mu\alpha\,.\,\top$ is the supertype of all recursive types

- Algorithmic formulation of disjointness, for example:

$$\frac{\Gamma \vdash \sigma_2 * \tau_2}{\Gamma \vdash \tau_1 \to \sigma_1 * \tau_2 \to \sigma_2} \qquad \frac{\Gamma, \alpha \vdash \sigma * \tau}{\Gamma \vdash \mu\alpha\,.\,\tau * \mu\alpha\,.\,\sigma}$$

- Our disjointness rules are sound w.r.t the specification

Completeness of disjointness

- Two recursive types $\mu\alpha . \tau$ and $\mu\alpha . \sigma$ satisfy the specification implies
- For any type ρ, if $\Gamma \vdash \mu\alpha . \tau \leq \rho$ and $\Gamma \vdash \mu\alpha . \sigma \leq \rho$ then ρ is top-like type

- By the disjointness rules for recursive types, we want to prove
- For any type ϑ, if $\Gamma, \alpha \vdash \tau \leq \vartheta$ and $\Gamma, \alpha \vdash \sigma \leq \vartheta$ then ϑ is top-like type

- ρ is the supertype of two recursive types
- ϑ is the supertype of two bodies of recursive types
- What is the relation between ρ and ϑ ?

Lower common supertype

- We define a function $\sqcup$ to compute a lower supertype of type τ and σ
- Function $\sqcup$ computes a common suppertype for two types whose contravariant positions are all $\bot$
- For example,

$$\tau_1 \to \sigma_1 \sqcup \tau_2 \to \sigma_2 = \bot \to (\sigma_1 \sqcup \sigma_2) \qquad \mu\alpha\,.\,\tau \sqcup \mu\alpha\,.\,\sigma = \mu\alpha\,.\,(\tau \sqcup \sigma)$$

- Two properties:
  - For any τ and σ, $\Gamma \vdash \tau \leq \tau \sqcup \sigma$ and $\Gamma \vdash \tau \leq \sigma \sqcup \sigma$
  - If $\Gamma \vdash \tau \leq \rho$ and $\Gamma \vdash \sigma \leq \rho$ and $\tau \sqcup \sigma$ is top-like then ρ is top-like

Informal proof for recursive case

1. We know for any ρ, if $\Gamma \vdash \mu\alpha\,.\,\tau \leq \rho$ and $\Gamma \vdash \mu\alpha\,.\,\sigma \leq \rho$ then ρ is top-like type
2. We assume that $\Gamma, \alpha \vdash \tau \leq \vartheta$ and $\Gamma, \alpha \vdash \sigma \leq \vartheta$
3. From the disjointness rule, for proving the goal, we need to prove ϑ is top-like type
4. We know that $\Gamma \vdash \mu\alpha\,.\,\tau \leq \mu\alpha\,.\,\tau \sqcup \mu\alpha\,.\,\sigma$ and $\Gamma \vdash \mu\alpha\,.\,\sigma \leq \mu\alpha\,.\,\tau \sqcup \mu\alpha\,.\,\sigma$
5. From (1) and (4), we know that $\mu\alpha\,.\,\tau \sqcup \mu\alpha\,.\,\sigma$ is a top-like type
6. According to the definition of lower common supertype, $\mu\alpha\,.\,(\tau \sqcup \sigma)$ is a top-like type
7. By inversion, $(\tau \sqcup \sigma)$ is a top-like type
8. Form (8) and (2), we claim that ϑ is top-like type

Summary

- In summary, the contributions of this paper are
  - Iso-recursive subtyping with intersection types
  - The λ_i^{μ} calculus and show it is type safety
  - Algorithmic disjointness for iso-recursive types
  - Mechanical formalization for all theorems

Thank you for your listening