IV. Subtyping in type theory

- Compare with set theory:

  $a \in A$ ------ $a : A$

  $A \subseteq B$ ------ $A \leq B$

  But, this is superficial, because typing and subtyping can only be more restrictive.

- Traditional notion: "subsumptive subtyping"

$$\frac{a : A \qquad A \leq B}{a : B}$$

Subtyping: motivations

- TTs in programming languages, eg,
  - More concise and readable programs (eg, subtype polymorphism in OO-languages)
- TTs for proof assistants, eg,
  - Abbreviations (eg, more readable terms/scripts)
- TTs as modelling languages, eg,
  - More powerful modelling languages

Subtyping: basics

- Fundamental principle

  If $A \leq B$ and, wherever a term of type B is required, we can use a term of type A instead.

  For example, the subsumption rule realises this.

- Basic laws
  - Reflexivity: $A \leq A$
  - Transitivity: $A \leq B$ & $B \leq C \Rightarrow A \leq C$

Subtyping: examples (informal)

- Base examples
  - $Nat \leq Int$
  - $Book \leq Phy$, where Phy is the type of physical objects
- Examples for type constructors
  - $A \leq A'$ & $B \leq B' \Rightarrow A \times B \leq A' \times B'$
  - $A' \leq A$ & $B \leq B' \Rightarrow A \to B \leq A' \to B'$ (contravariance)
  - $A \leq B \Rightarrow List(A) \leq List(B)$
  - $\langle l_1 : A_1, \ldots, l_n : A_n \rangle \leq \langle l_1 : A_1, \ldots, l_{n-1} : A_{n-1} \rangle$ (record types)

Question:

Is subsumptive subtyping adequate for type theories with canonical objects?

Answer:

No: why and then what?

Two different views of types

- Recall: Curry-style v.s. Church-style presentations
- Type assignment systems (TASs) v.s. TTs with canonical objects
  - Type assignment: Objects exist first and λ-terms are overloaded with more than one type. Eg, $\lambda x.x : \alpha \to \alpha$.

    Examples: type systems in PLs such as ML, Haskell, ...
  - Canonical objects: Types and their objects co-exist (one does not without the other) and a λ-term has unique typing, among others.

    Examples: Martin-Lof's TT, CIC, UTT, ...

Two corresponding views of subtyping

Views on types	Views on subtyping
type assignment systems	subsumptive subtyping
TTs with canonical objects	???

- Subsumptive subtyping is suitable for type assignment:
  - A term can be overloaded (has more than one type).
  - Subsumption is simply another rule for type assignment.
- What about TTs with canonical objects?

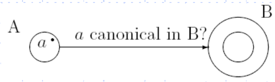
Incompatibility of subsumption & canonicity

- Subsumption rule:

$$\frac{a : A \quad A \le B}{a : B}$$

Incompatible with the view of canonical objects

Q: If A≤B and a:A is canonical in A, is it canonical in B?



Canonicity

- Definition

  Any closed object of an inductive type is computationally equal to a canonical object of that type.
- This is a basis of TTs with canonical objects.
  - This is why the elimination rule is adequate.
  - Eg, Elimination rule for List(T):

    "For any family C, if C is inhabited for all canonical T-lists nil(T) and cons(T,a,l), then so is C for all T-lists."

- Canonicity is lost in subsumptive subtyping.
  - Eg, $\dfrac{A \le B}{List(A) \le List(B)}$
  - nil(A) : List(B), by subsumption;
  - But nil(A) ≠ any canonical B-list nil(B) or cons(B,b,l).
  - The elim rule for List(B) is inadequate: it does not cover nil(A) … …

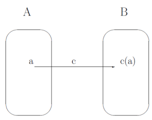
Amending the elimination rule?

- Generalise it to cover all subtypes …
  - to take care of the objects introduced by subsumptive subtyping.
- But
  - This requires "bounded quantification" to quantify over all subtypes (of the form ∀A≤B …)
  - Troublesome …
- If not, then what?

$$\begin{aligned}\mathcal{E}'_{List} \quad : \quad & (A:Type)(A_0 \le A:Type)\\ & (C:(List(A_0))Type)\\ & (c:C(nil(A_0)))\\ & (f:(a:A_0)(l:List(A_0))(C(l))C(cons(A_0,a,l)))\\ & (z:List(A_0))C(z)\end{aligned}$$

Coercive subtyping

- Basic idea
  - A≤B if there is a coercion c from A to B:

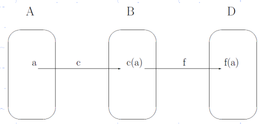


  - Coercions are "implicit" – they can be omitted!
- Subtyping as abbreviation

Coercive subtyping: formal rules

- Formal presentation (Luo 1997/1999) includes

$$\frac{f : B \to D \qquad a : A \qquad A \leq_c B}{f(a) = f(c(a)) : D}$$



a : A $\leq_c$ B ➔ "a" can be regarded as an object of type B
➔ $\mathbf{f}_B[a] = \mathbf{f}_B[c(a)]$, ie, "a" stands for "c(a)"

Coherence

- Coherence: a key requirement
  - Coercions between any two types are unique.
  - Think of an implementation: if more than one, the computer does not know which to choose ...
  - Incoherence leads to non-conservativity (and in most cases, inconsistency).
- Formal defn of coherence:

$$\frac{A <_c B \qquad A <_{c'} B}{c = c' : A \to B}$$

where = is the computational equality.

Coercive subtyping

- Adequate for TTs with canonical objects
  - Eg, Martin-Löf's TT, CIC and UTT
- Solves the incompatibility problem of subsumption
- Preserves good meta-theoretic properties (Soloviev & Luo 2002)
  - Canonicity, Normalisation, Logical consistency, … …
  - Work by Xue on a new conservativity proof (TYPES 2010)
- Coercions in proof assistants
  - Coq (Saïbi 1997), Lego (Bailey 1998), Plastic (Callaghan & Luo 2001), Matita (2008)

- Coercive subtyping generalises/subsumes
  - Injective/subset subtyping: Even ≤ N; Man ≤ Human.
  - Projective/inheritance subtyping: Σ(Man, handsome) ≤ Man.
- Applications of coercive subtyping, including
  - Proof development
  - Dependently-typed programming
  - Type-theoretical semantics in linguistics
  - … …

Structural subtyping in coercive subtyping

- Example – structural subtyping for lists:

$$\frac{A \leq_c B}{List(A) \leq_{map(c)} List(B)}$$

- Structural subtyping for all inductive types
  - Σ-types, types of vectors, …
  - General rules and transitivity elimination [Luo & Adams 08, Luo & Luo 05]

Non-structural subtyping – examples

- Projective subtyping (c.f. record subtyping)
  - From Σ-types or record types to component types
    - First projection as a coercion: $\Sigma(Nat, positive) \leq_{\pi_1} Nat$
    - Projections of (dependent) record types
  - Very useful in proofs/modelling
  - Eg,
    - Proof development [Bailey 1998, …]
    - Type-theoretic model of linguistic semantics [Luo 2010]

Non-structural subtyping – examples (cont'd)

- Coercion ξ concerning unit types

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{1}(A,a) \leq_{\xi_{A,a}} A : Type}$$

where $\xi_{A,a}(x) = a$ for any $x : \mathbf{1}(A,a)$.

  - Useful in various applications
  - Eg, representation of manifest fields in module types (Σ-types or dependent record types) [Luo 08]

Coercive subtyping is

an adequate theory of subtyping
for type theories with canonical objects.

Views on types	**Views on subtyping**
type assignment systems	subsumptive subtyping
TTs with canonical objects	coercive subtyping

Historical remarks

- Semantic interpretations for subtyping in PLs
  - Early papers: eg,
    - (Mitchell 1983/1991) for the simply typed λ-calculus
    - Both subsumptive and coercive interpretations, called subset interpretation and coercion interpretation, resp.
  - Later papers, eg,
    - (Breazu-Tannen et al 1991) for recursive & record types in PLs.
- Subsumptive subtyping for dependent types
  - Subtyping for Edinburgh LF (Aspinall & Campagnoni 2001)
- Remarks on our previous treatment in coercive subtyping
  - Proof-theoretic
  - For TTs with canonical objects

Revision Questions

- How can subtyping be useful in
  - Programming languages?
  - Proof assistants?
  - Modelling?
- What are two different views of types/subtyping?
  - What is subsumptive subtyping?
  - What is coercive subtyping?
  - How are they related to the views of types?

Selected References

- D. Aspinall and A. Compagnoni. Subtyping dependent types. Theoretical Computer Science, 266(1-2). 2001.
- A. Bailey. The Machine-checked Literate Formalisation of Algebra in Type Theory. PhD thesis, University of Manchester, 1999.
- V. Breazu-Tannen, T. Coquand, C. Gunter and A. Scedrov. Inheritance and explicit coercion. Information and Computation, 93. 1991.
- P. Callaghan and Z. Luo. An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning, 27(1). 2001.
- Z. Luo. Coercive subtyping in type theory. CSL'96, LNCS 1258. 1996.
- Z. Luo. Coercive subtyping. J. of Logic and Computation, 9(1). 1999.
- J. Mitchell. Type inference with simple subtypes. J. of Functional Programming, 1(3). 1991.
- A. Saïbi. Typing algorithm in type theory with inheritance. POPL'97, 1997.