Computation and Deduction

Lecture 28: Atomic Subtyping
April 29, 1997

1. λ-Calculus with Subtyping
2. Algorithmic Subtyping and Type-Checking
3. Soundness and Completeness
4. Compiling to Coercions

λ-Calculus with Atomic Subtyping I

```
%%% Built-in types, unspecified
integer : type.                          %name integer I
real_number : type.                      %name real_number R

%%% Atomic types
atom : type.                             %name atom P

int : atom.
real : atom.

%%% Types
tp : type.                               %name tp A

atm : atom -> tp.
=> : tp -> tp -> tp.                     %infix right 11 =>
```

λ-Calculus with Atomic Subtyping II

```
%%% Terms
tm : type.                                    %name tm M

iii : integer -> tm.
rrr : real_number -> tm.

lam : tp -> (tm -> tm) -> tm.
app : tm -> tm -> tm.
```

Declarative Subtyping

```
%%% Atomic Subtyping
<< : atom -> atom -> type.                %name << AS
                                          %infix none 10 <<
int<<real : int << real.

%%% Subtyping
< : tp -> tp -> type.                     %name < S
                                          %infix none 10 <
<refl : A < A.
<trans : A < B -> B < C -> A < C.
<atm : P << Q
        -> atm P < atm Q.
<arrow : B1 < A1
          -> A2 < B2
          -> (A1 => A2) < (B1 => B2).
```

Declarative Typing

```
of : tm -> tp -> type.                          %name of P

of_sub : of M A
          -> A < B
          -> of M B.

of_iii : of (iii I) (atm int).
of_rrr : of (rrr R) (atm real).

of_lam : ({x:tm} of x A -> of (M x) B)
          -> of (lam A M) (A => B).

of_app : of M (B => A)
          -> of N B
          -> of (app M N) A.
```

Algorithmic Subtyping

```
%%% Algorithmic atomic subtyping
<<! : atom -> atom -> type.                %name <<! AG
                                           %infix none 10 <<!

int<<!int  : int <<! int.
int<<!real : int <<! real.
real<<!real : real <<! real.

%%% Algorithmic subtyping
<! : tp -> tp -> type.                     %name <! G
                                           %infix none 10 <!

<!atm : atm P <! atm Q
         <- P <<! Q.

<!arrow : (A1 => A2) <! (B1 => B2)
           <- B1 <! A1
           <- A2 <! B2.
```

Algorithmic Type-Checking

```
%%% Algorithmic type checking
of! : tm -> tp -> type.                          %name of! Q

of!_iii : of! (iii I) (atm int).

of!_rrr : of! (rrr R) (atm real).

of!_lam : of! (lam A M) (A => B)
           <- ({x:tm} of! x A -> of! (M x) B).

of!_app : of! (app M N) A
           <- of! M (B => A)
           <- of! N B'
           <- B' <! B.
```

Soundness of Algorithmic Subtyping

```
%%% Atomic subtyping
sound<<! : P <<! Q -> (atm P) < (atm Q) -> type.

sd<<!_nn : sound<<! (int<<!int) (<refl).
sd<<!_nr : sound<<! (int<<!real) (<atm int<<real).
sd<<!_rr : sound<<! (real<<!real) (<refl).

%%% Subtyping
sound<!  : A <! B -> A < B -> type.

sd<!_atm : sound<! (<!atm G) S
            <- sound<<! G S.

sd<!_arrow : sound<! (<!arrow G2 G1) (<arrow S1 S2)
              <- sound<! G1 S1
              <- sound<! G2 S2.
```

Soundness of Algorithmic Type-Checking

```
%%% Type-checking
sound!   : of! M A -> of M A -> type.

sd!_int : sound! (of!_iii) (of_iii).
sd!_real : sound! (of!_rrr) (of_rrr).

sd!_lam : sound! (of!_lam Q) (of_lam P)
           <- ({x:tm} {q:of! x A} {p:of x A}
                 sound! q p
                 -> sound! (Q x q) (P x p)).

sd!_app : sound! (of!_app G Q2 Q1)
                 (of_app P1 (of_sub P2 S))
           <- sound<! G S
           <- sound! Q1 P1
           <- sound! Q2 P2.
```

Algorithmic Subtyping, Reflexivity

```
lemma1a : {P:atom} P <<! P -> type.
lm1a_int : lemma1a int (int<<!int).
lm1a_real : lemma1a real (real<<!real).

lemma1b : {A:tp} A <! A -> type.

lm1b_atm : lemma1b (atm P) (<!atm AG)
            <- lemma1a P AG.

lm1b_arrow : lemma1b (A1 => A2) (<!arrow G2 G1)
              <- lemma1b A1 G1
              <- lemma1b A2 G2.
```

Algorithmic Subtyping, Transitivity

```
lemma2a : P <<! Q -> Q <<! R -> P <<! R -> type.

lm2a_iiii : lemma2a (int<<!int) (int<<!int) (int<<!int).
lm2a_iiir : lemma2a (int<<!int) (int<<!real) (int<<!real).
lm2a_irrr : lemma2a (int<<!real) (real<<!real) (int<<!real).
lm2a_rrrr : lemma2a (real<<!real) (real<<!real) (real<<!real).

lemma2b : A <! B -> B <! C -> A <! C -> type.

lm2b_atm : lemma2b (<!atm AG) (<!atm AH) (<!atm AI)
            <- lemma2a AG AH AI.

lm2b_arrow : lemma2b (<!arrow G2 G1) (<!arrow H2 H1)
                     (<!arrow I2 I1)
              <- lemma2b H1 G1 I1
              <- lemma2b G2 H2 I2.
```

Completeness of Atomic Subtyping

```
comp<<! : P << Q -> P <<! Q -> type.

comp<<!_ir : comp<<! (int<<real) (int<<!real).
```

Completeness of Subtyping

```
comp<! : A < B -> A <! B -> type.

comp<!_refl : comp<! (<refl : A < A) G
               <- lemma1b A G.

comp<!_trans : comp<! (<trans S1 S2) G3
                <- comp<! S1 G1
                <- comp<! S2 G2
                <- lemma2b G1 G2 G3.

comp<!_atm : comp<! (<atm AS) (<!atm AG)
              <- comp<<! AS AG.

comp<!_arrow : comp<! (<arrow S1 S2) (<!arrow G2 G1)
                <- comp<! S1 G1
                <- comp<! S2 G2.
```

Completeness of Type-Checking

```
comp! : of M A -> of! M B -> B < A -> type.

comp!_sub : comp! (of_sub P1 S) Q1 (<trans S1 S)
             <- comp! P1 Q1 S1.

comp!_iii : comp! (of_iii) (of!_iii) (<refl).

comp!_rrr : comp! (of_rrr) (of!_rrr) (<refl).

comp!_lam : comp! (of_lam P1) (of!_lam Q1) (<arrow <refl S1)
            <- ({x:tm} {p:of x A} {q:of! x A}
                  comp! p q (<refl)
                  -> comp! (P1 x p) (Q1 x q) S1).

comp!_app : comp! (of_app P1 P2) (of!_app G1 Q2 Q1) S12
            <- comp! P1 Q1 (<arrow S11 S12)
            <- comp! P2 Q2 S2
            <- comp<! (<trans S2 S11) G1.
```

λ-Calculus with Coercions

```
obj : tp -> type. %name obj E

ii : integer -> obj (atm int).
rr : real_number -> obj (atm real).

ir : obj (atm int) -> obj (atm real).

lm : (obj A -> obj B) -> obj (A => B).
ap : obj (A => B) -> obj A -> obj B.
```

Compiling Atomic Subtyping to Coercions

```
compile<< : P << Q -> (obj (atm P) -> obj (atm Q)) -> type.

compile<<ir : compile<< (int<<real) (ir).
```

Compiling Subtyping to Coercions

```
compile< : A < B -> (obj A -> obj B) -> type.

compile<refl : compile< (<refl) ([x:obj A] x).

compile<trans : compile< (<trans S1 S2)
                         ([x:obj A] C2 (C1 x))
                  <- compile< S1 C1
                  <- compile< S2 C2.

compile<atm : compile< (<atm AS) C
                <- compile<< AS C.

compile<arrow : compile< (<arrow S1 S2)
                  ([x:obj (A1 => A2)]
                     lm [y:obj B1] C2 (ap x (C1 y)))
                  <- compile< S1 C1
                  <- compile< S2 C2.
```

Compiling Typing Derivations

```
compile : of M A -> obj A -> type.

compile_sub : compile (of_sub P S) (C E)
               <- compile P E
               <- compile< S C.

compile_iii : compile (of_iii : of (iii I) (atm int)) (ii I).
compile_rrr : compile (of_rrr : of (rrr R) (atm real)) (rr R).

compile_lam : compile (of_lam P) (lm E)
               <- ({x:tm} {p:of x A} {e:obj A}
                     compile p e
                     -> compile (P x p) (E e)).

compile_app : compile (of_app P1 P2) (ap E1 E2)
               <- compile P1 E1
               <- compile P2 E2.
```