Computation and Deduction

Lecture 8: Parametric and Hypothetical Judgments
February 6, 1997

1. Mini-ML Expressions
2. Evaluation Judgment
3. Value Judgment
4. Value Soundness
5. Sample Queries
6. Typing Judgment
7. Static and Dynamic Types

Mini-ML Expressions

```
exp  : type.  %name exp E

z    : exp.
s    : exp -> exp.
case : exp -> exp -> (exp -> exp) -> exp.
pair : exp -> exp -> exp.
fst  : exp -> exp.
snd  : exp -> exp.
lam  : (exp -> exp) -> exp.
app  : exp -> exp -> exp.
letv : exp -> (exp -> exp) -> exp.
letn : exp -> (exp -> exp) -> exp.
fix  : (exp -> exp) -> exp.
```

Mini-ML Types

```
tp : type.  %name tp T

nat   : tp.
cross : tp -> tp -> tp.
arrow : tp -> tp -> tp.
```

Evaluation Judgment I

```
eval : exp -> exp -> type.  %name eval D

% Natural Numbers
ev_z       : eval z z.
ev_s       : eval (s E) (s V)
               <- eval E V.
ev_case_z  : eval (case E1 E2 E3) V
               <- eval E1 z
               <- eval E2 V.
ev_case_s  : eval (case E1 E2 E3) V
               <- eval E1 (s V1')
               <- eval (E3 V1') V.
```

Evaluation Judgment II

```
% Pairs
ev_pair : eval (pair E1 E2) (pair V1 V2)
             <- eval E1 V1
             <- eval E2 V2.
ev_fst  : eval (fst E) V1
             <- eval E (pair V1 V2).
ev_snd  : eval (snd E) V2
             <- eval E (pair V1 V2).
```

Evaluation Judgment III

```
% Functions
ev_lam  : eval (lam E) (lam E).
ev_app  : eval (app E1 E2) V
            <- eval E1 (lam E1')
            <- eval E2 V2
            <- eval (E1' V2) V.

% Definitions
ev_letv : eval (letv E1 E2) V
            <- eval E1 V1
            <- eval (E2 V1) V.
ev_letn : eval (letn E1 E2) V
            <- eval (E2 E1) V.

% Recursion
ev_fix  : eval (fix E) V
            <- eval (E (fix E)) V.
```

Value Judgment

```
value : exp -> type.  %name value P

val_z     : value z.
val_s     : value (s E)
             <- value E.
val_pair  : value (pair E1 E2)
             <- value E1
             <- value E2.
val_lam   : value (lam E).
```

Value Soundness I

```
vs : eval E V -> value V -> type.

% Natural Numbers
vs_z      : vs (ev_z) (val_z).

vs_s      : vs (ev_s D1) (val_s P1)
             <- vs D1 P1.

vs_case_z : vs (ev_case_z D2 D1) P2
             <- vs D2 P2.

vs_case_s : vs (ev_case_s D3 D1) P3
             <- vs D3 P3.
```

Value Soundness II

```
% Pairs
vs_pair : vs (ev_pair D2 D1) (val_pair P2 P1)
            <- vs D1 P1
            <- vs D2 P2.

vs_fst  : vs (ev_fst D') P1
            <- vs D' (val_pair P2 P1).

vs_snd  : vs (ev_snd D') P2
            <- vs D' (val_pair P2 P1).
```

Value Soundness III

```
% Functions
vs_lam  : vs (ev_lam) (val_lam).

vs_app  : vs (ev_app D3 D2 D1) P3
           <- vs D3 P3.

% Definitions
vs_letv : vs (ev_letv D2 D1) P2
           <- vs D2 P2.

vs_letn : vs (ev_letn D2) P2
           <- vs D2 P2.

% Recursion
vs_fix : vs (ev_fix D1) P1
          <- vs D1 P1.
```

Sample Queries

```
solve* 1
D : eval (app (lam [x:exp] x) z) V.%.
Using: eval.elf value.elf ...
Solving for: eval value ...
Solving...

V = z,
D = ev_app ev_z ev_z ev_lam.
yes
%% OK %%

solve* 4
D : eval (fst z) V.
Solving...
no
%% OK %%
```

Error Messages

```
solve* 1
ev_app ev_lam ev_z ev_z : eval E V.

std_in:1.21-1.25 Error: Type checking failed
ev_z  : eval z z <> eval z (lam E1)
Unification failure due to clash: lam <> z

%% ERROR %%  TypeCheckFail
%% ABORT %%
```

Executing a Proof

```
% generating D
D : eval (app
            (fix [f:exp] lam [x:exp]
               (case x z ([x':exp] s (s (app f x')))))
            (s z))
     V.
D = ...,
V = s (s z).

% value soundness on D
vs (ev_app
      (ev_case_s
          (ev_s (ev_s (ev_app (ev_case_z ev_z ev_z)
                              ev_z (ev_fix ev_lam))))
          (ev_s ev_z))
      (ev_s ev_z) (ev_fix ev_lam))
   P.
P = val_s (val_s val_z).
(( ...
   case E21 z E33 = E1 z,
   ... ))
```

Sigma "Types"

```
sigma [D: eval (app (fix [f] lam [x]
          (case x z ([x':exp] s (s (app f x')))))
          (s z)) V]
 vs D P.

P = val_s (val_s val_z),
V = s (s z).
```

Typing Judgment I

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

% Natural Numbers
tp_z     : of z nat.

tp_s     : of (s E) nat
             <- of E nat.

tp_case  : of (case E1 E2 E3) T
             <- of E1 nat
             <- of E2 T
             <- ({x:exp} of x nat -> of (E3 x) T).
```

Typing Judgment II

```
% Pairs
tp_pair : of (pair E1 E2) (cross T1 T2)
           <- of E1 T1
           <- of E2 T2.
tp_fst  : of (fst E) T1
           <- of E (cross T1 T2).
tp_snd  : of (snd E) T2
           <- of E (cross T1 T2).
```

Typing Judgment IIO

```
% Functions
tp_lam : of (lam E) (arrow T1 T2)
          <- ({x:exp} of x T1 -> of (E x) T2).
tp_app : of (app E1 E2) T1
          <- of E1 (arrow T2 T1)
          <- of E2 T2.

% Definitions
tp_letv : of (letv E1 E2) T2
           <- of E1 T1
           <- ({x:exp} of x T1 -> of (E2 x) T2).
tp_letn : of (letn E1 E2) T2
           <- of E1 T1
           <- of (E2 E1) T2.

% Recursion
tp_fix : of (fix E) T
          <- ({x:exp} of x T -> of (E x) T).
```

Sample Queries

CONFIG file:

```
-static mini-ml.elf
-static tp.elf
-dynamic tpinf.elf
-query examples.quy
```

```
Q : of (lam [x] pair x (s x)) T.
Solving...
T = arrow nat (cross nat nat),
Q = tp_lam [x:exp] [P:of x nat] tp_pair (tp_s P) P.
%% OK %%
of (lam [x] x) T.
Solving...

T = arrow T1 T1.
```

Dynamic Families

CONFIG2 file:

```
-static mini-ml.elf
-dynamic tp.elf
-dynamic tpinf.elf
-query examples3.quy
```

```
solve* 2
of (lam [x] x) T.
Using: tp.elf tpinf.elf
Solving for: tp of
Solving...

T = arrow nat nat.
T = arrow (cross nat nat) (cross nat nat).
```

Static Families

CONFIG3:

```
-static mini-ml.elf
-static tp.elf
-static tpinf.elf
-query examples3.quy
```

```
solve* 1
Q : of (lam [x] x) T.
Using:
Solving for:
Solving...

T = T,
Q = Q.
yes
```