Linear Logic

Lecture 08: Unification
February 5, 1998

1. Existential variables
2. Unification
3. Extended Occurs-check
4. Trailing

Terms with Existential Variables

```
 datatype term =                                  (* t ::=          *)
  Parm of parm                                    (*   a            *)
| Var of var                                      (* | x            *)
| Root of const * term list                       (* | f(t1,...,tn) *)
| EVar of parm Ctx.ctx * (term option) ref (* | X^P<=I>     *)

val newEVar : parm Ctx.ctx -> term

exception Unify of string

val unify : term * term -> unit       (* may raise Unify *)
val unify' : term list * term list -> unit (* may raise Unify *)
```

Unification I

```
(* unify (t, s) => ()      t, s closed
   unify' (ts, ss) => ()  ts, ss closed
   Effect: instantiate EVars
           raises Unify (reason) if not unifiable
*)
fun unify (Parm(a), Parm(a')) =
    if a = a' then ()
    else raise Unify ("Parameter clash")
  | unify (Root(f, ts), Root(g, ss)) =
    if f = g then unify' (ts, ss)
    else raise Unify ("Constant clash")
  | unify (EVar(_, ref(SOME(t))), s) = unify (t, s)
  | unify (t, EVar(_, ref(SOME(s)))) = unify (t, s)
```

Unification II

```
  | unify (X as EVar(_, r), X' as EVar(_, r')) =
    if r = r' then () else unifyEVar (X, X')
  | unify (X as EVar _, s) = unifyEVar (X, s)
  | unify (t, X as EVar _) = unifyEVar (X, t)
  (* Var(x) on either side impossible *)

and unify' (nil, nil) = ()
  | unify' (t::ts, s::ss) = (unify (t, s); unify' (ts, ss))
  | unify' _ = raise Unify ("Argument number")

and unifyEVar (X as EVar (P, r), t) =
      (check (X, t); instantiate (r, t))
```

Extended Occurs-Check

```
fun check (EVar(P, r), t) =
    let fun chk (Parm(a)) =
            if member (a, P) then ()
            else raise Unify ("Parameter occurrence")
          | chk (Root(f, ts)) = chk' ts
          | chk (EVar (P', ref(SOME(t)))) = chk t
          | chk (EVar (P', r')) =
            if r = r' then raise Unify ("Variable occurrence")
            else if contained (P', P) then ()
                 else instantiate (r', newEVar (restrict (P', P)))
          (* Var(x) impossible *)
        and chk' (nil) = ()
          | chk' (t::ts) = (chk t; chk' ts)
    in
      chk t
    end
```

Trailing

```
fun newEVar (P) = EVar (P, ref(NONE))

local
  val localTrail = ref (nil : term option ref list)

  fun unwind (nil) = ()
    | unwind (r::tr') = (r := NONE; unwind tr')
in
  fun instantiate (r, t) =
      (r := SOME(t);
       localTrail := (r::(!localTrail)))
  fun trail f x =
      let val r = f x handle exn => (unwind (!localTrail);
                                     localTrail := nil;
                                     raise exn)
          val _ = (localTrail := nil)
      in r end
end
```