Linear Logic

Lecture 07: Proof Search and Inversion
February 3, 1998

1. Data Structures
2. Rules
3. Top Level

Contexts

```
signature CTX =
sig

  datatype 'a ctx =
    Null
  | $ of 'a ctx * 'a

  (* infix $ *)

  (* nth (Null $ xk $ ... $ x1, n) = xn *)
  (* raises Subscript if n > k *)
  val nth : 'a ctx * int -> 'a

  val member : ''a * ''a ctx -> bool

end;  (* signature CTX *)
```

Terms

```
signature TERM =
sig
  structure Ctx : CTX

  type const = string                        (* Constants c,f  *)
  type parm = string                         (* Parameters a   *)
  type var = string                          (* Variables x    *)

                                             (* Terms          *)
  datatype term =                            (* t ::=          *)
    Parm of parm                             (*   a            *)
  | Var of var                               (* | x            *)
  | Root of const * term list                (* | f(t1,...,tn) *)

  (* subst (t,x) s ===> [t/x] s,  t closed *)
  val subst : (term * var) -> term -> term
```

```
  (* eq (t, s) ===> true  iff  t = s, t and s closed *)
  val eq : term * term -> bool
  and eq' : term list * term list -> bool

  (* closed P t ===> true  iff t closed wrt P *)
  val closed : parm Ctx.ctx -> term -> bool

  (* legal a ===> true  iff a is a legal external parameter *)
  (* External parameters are not allows to start with a digit. *)
  val legal : parm -> bool

  (* newParm () ===> a,  where a is a new internal parameter *)
  (* Internal parameters are natural numbers. *)
  val newParm : unit -> parm

  val toString : term -> string

end;  (* signature TERM *)
```

Propositions

```
signature PROP =
sig
                                          (* Propositions   *)
  datatype prop =                         (* A ::=          *)
    Atom of string * Term.term list       (*   P(t1,...,tn) *)
  | Tensor of prop * prop                 (* | A1 * A2      *)
  | One                                   (* | 1            *)
  | Lolli of prop * prop                  (* | A1 -o A2     *)
  | With of prop * prop                   (* | A1 & A2      *)
  | Top                                   (* | T            *)
  | Plus of prop * prop                   (* | A1 + A2      *)
  | Zero                                  (* | 0            *)
  | Implies of prop * prop                (* | A1 => A2     *)
  | Bang of prop                          (* | ! A          *)
  | Forall of Term.var * prop             (* | forall x. A  *)
  | Exists of Term.var * prop             (* | exists x. A  *)
```

```
  (* subst (t,x) A ===> [t/x] A; t closed *)
  val subst : Term.term * Term.var -> prop -> prop

  (* eq (A, B) ===> true  iff  A = B (modulo variable names); *)
  (* A and B closed *)
  val eq : prop * prop -> bool

  (* closed P A ===> true  iff  A closed wrt P *)
  val closed : Term.parm Ctx.ctx -> prop -> bool

  val toString : prop -> string

end;  (* signature PROP *)
```

Sequents

```
signature SEQ =
sig
  (*
     A sequent P; G; D ==> C consists of
     a parameter context P which contains all individual parameters
       which may occur in G, D, and C,
     a context G of unrestricted hypotheses,
     a context D of linear hypothesis,
     and a proposition C.
  *)
  type seq = Prop.Term.parm Ctx.ctx     (* [P]   *)
             * Prop.prop Ctx.ctx        (* G;    *)
             * Prop.prop Ctx.ctx        (* D     *)
             * Prop.prop                (* ==> C *)

  val toString : seq -> string
```

```
  (*
     We maintain the invariant that Success does not appear as a
     conjunct in a goal structure.
  *)
  datatype goal =
    Success
  | Conj of goal * goal
  | Seq of seq

  exception Fail of string

  (* A rule reduces a sequent to subgoals *)
  (* Currently we do not attempt to validate inferences! *)
  type rule = seq -> goal

end;  (* signature SEQ *)
```

Rules

```
signature RULES =
sig
  type hyp = int                              (* Hypothesis H *)
  type hyps = hyp list                        (* Hypotheses Hs *)
  type split = hyps * hyps                    (* Split (Hs1, Hs2) *)

  val initial : Seq.rule
  val derelict : hyp -> Seq.rule

  val tensorR : split -> Seq.rule
  val tensorL : hyp -> Seq.rule

  val lolliR : Seq.rule
  val lolliL : split -> hyp -> Seq.rule
  ...
```

```
  (* forallR a : rule; a new wrt parameter context P *)
  val forallR : Seq.Prop.Term.parm -> Seq.rule

  (* forallL t H : rule; t closed wrt parameter context P *)
  val forallL : Seq.Prop.Term.term -> hyp -> Seq.rule

  (* existsR t : rule; t closed wrt parameter context P *)
  val existsR : Seq.Prop.Term.term -> Seq.rule

  (* existsL a H : rule; a new wrt parameter context P *)
  val existsL : Seq.Prop.Term.parm -> hyp -> Seq.rule

end;  (* signature RULES *)
```

Top Level

```
signature TOP =
sig
  (* Propositions A and B must be closed *)
  val prove : Seq.Prop.prop -> unit     (* [.] .; . |- A ? *)
  val entail : Seq.Prop.prop * Seq.Prop.prop -> unit
                                        (* [.] .; A |- B ? *)

  val show : unit -> unit               (* current goal sequent *)
  val showAll : unit -> unit            (* all current subgoals *)
  val apply : Seq.rule -> unit          (* apply rule to current goal *)
  val judgment : unit -> bool           (* print evident judgment *)
  val done : unit -> unit               (* check if judgment is evident *)
end; (* signature TOP *)
```