THE Kleisli QUERY SYSTEM
Reference Manual

Limsoon Wong

Institute of Systems Science
Heng Mui Keng Terrace, Singapore 0511
email: limsoon@iss.nus.sg

24 November 1997

Version D Release 24 November 1997

Contents

1 Introduction

1.1 Overview

Based on some ideas described in Buneman et. al. [6], I have built a prototype query system called Kleisli. The system is designed as a database engine to be connected to the host programming language ML [14] via a collection of libraries routines. These routines are parameterized for more open and better control so that expert users do not have to resort to wily evasion of restriction in their quest for performance. (There is strong evidence [20] a canny persistence in uncovering necessary detail to satisfy their concern for performance.)

I have included an implementation of a high-level query language, for non-expert users, call CPL, with the prototype. CPL stands for *Collection Programming Language.* The libraries actually contain enough tools for a competent user to quickly build his own query language or command line interpretor to use in connection with Kleisli and the host language ML. In fact, CPL is an example of how to use these tools to implement query languages for Kleisli. This report documents the modules of Kleisli.

1.2 Pre-requisites

Kleisli and CPL are implemented on top of the function programming language ML; in particular, we use version 093 of the Standard ML of New Jersey.

For general information on functional programming, see Field and Harrison [7]. For an introduction to ML, see Paulson [19]. For specifics of Standard ML, consult Milner et. al. [14].

1.3 Organization

1.4 Notations

This manual involves three "sytems": Standard ML, Kleisli, and CPL. So we need some notations to improve clarity.

San-serif font is used for names of modules and functions of the Kleisli query system. `Type-writer` font is used for names of primitives of CPL, as well as fragments of source codes written in SML. *Italic* font is used for meta symbols.

Complex objects manipulated by Kleisli are implemented as SML objects and always have the SML type CO.CO, regardless of their actual types in CPL. But whenever possible, instead of writing CO.CO, we write down the actual CPL types. To avoid confusing the two, we enclose CPL types between slashes. For example, we write `/{num}/` instead of CO.CO. Consult the *Collection Programming Language Reference Manual* [24] for a description of CPL types.

1.5 Acknowledgements

I would like to thank Kyle Hart and Jonathan Crabtree for developing the mediators between Kleisli and Sybase and between Kleisli and GenBank. I would also like to thank Peter Buneman, Susan Davidson, and Chris Overton for inspiring this project. Last but not least, Leonid Libkin, Dan Suciu, and Val Tannen also greatly influenced this project.

Part I

Complex Object Interface

2 Introduction

3 CO

3.1 Synopsis

structure CO

This module defines the structure of complex objects.

The structure given here can only be used in doing pattern matching when programming with Kleisli in ML. Complex objects should *not* be constructed directly using the constructors given in this module — use the various Mk operations provided in CONum, COSet, and so on instead.

3.2 Signature

```
type 'a LAZY = unit ->'a
type tno = int (* user-defined base type number *)
type lno = int (* label number *)

datatype CO
  = Integer    of int
  | Real       of real
  | Bool       of bool
  | String     of string
  | Unit
  | User       of tno * CO
  | EmptySet
  | SngSet     of CO
  | UnionSet   of CO * CO
  | lUnionSet  of CO LAZY * CO LAZY
  | EmptyBag
  | SngBag     of CO
  | UnionBag   of CO * CO
  | lUnionBag  of CO LAZY * CO LAZY
  | EmptyList
  | SngList    of CO
  | UnionList  of CO * CO
  | lUnionList of CO LAZY * CO LAZY
  | Record     of CO Record.record
  | Variant    of lno * CO
  | Fun        of CO -> CO


val IsRecord          : CO -> bool
val IsVariant         : CO -> bool
val IsUser            : CO -> bool
val IsUser'           : tno -> CO -> bool
val IsBool            : CO -> bool
val IsString          : CO -> bool
```

```
val IsNum             : CO -> bool
val IsUnit            : CO -> bool
val IsSysBase         : CO -> bool
val IsBase            : CO -> bool
val IsSet             : CO -> bool
val IsBag             : CO -> bool
val IsList            : CO -> bool
val IsCollection      : CO -> bool
val IsEqType          : CO -> bool
val IsPotentialEqType : CO -> bool
val IsFun             : CO -> bool

val Sri     : (CO.CO -> 'a -> 'a) -> 'a -> CO.CO -> 'a
val Sru     : ('a -> 'a -> 'a) -> (CO.CO -> 'a) -> 'a -> CO.CO -> 'a
val Listify : CO.CO -> CO.CO list
```

3.3 Description

- CO

  The complex object type used by Kleisli. The representation given here is for pattern matching in ML only. Do not use it to directly construct complex objects. Note the presence of lUnionSet, lUnionBag, and lUnionList which directly incorporate coarse-grained laziness into the representation of complex objects. Note also the use of union presentation for sets, bags, and lists.

- IsRecord M

  True iff M is a complex object of record type.

- IsVariant M

  True iff M is a complex object of variant type.

- IsUser M

  True iff M is a complex object of user-defined type.

- IsUser' n M

  True iff M is a complex object of user-defined type n.

- IsBool M

  True iff M is a complex object of Boolean type.

- IsString M

  True iff M is a complex object of string type.

- IsNum M

  True iff M is a complex object of numeric type.

- IsUnit M

  True iff M is a complex object of unit type.

- $\mathsf{IsSysBase}\ M$

  True iff M is a complex object of system-defined base type (ie., Boolean, numeric, string, or unit).

- $\mathsf{IsBase}\ M$

  True iff M is a complex object of base type (ie., system-defined base type or user-defined base type).

- $\mathsf{IsSet}\ M$

  True iff M is a complex object of set type.

- $\mathsf{IsBag}\ M$

  True iff M is a complex object of bag type.

- $\mathsf{IsList}\ M$

  True iff M is a complex object of list type.

- $\mathsf{IsCollection}\ M$

  True iff M is a complex object of set, bag, or list type.

- $\mathsf{IsEqType}\ M$

  True iff M is a complex object which has a computable equality test. Hence M must either be a collection or a base type object.

- $\mathsf{Is\ PotentialEqType}\ M$

  True iff M is a complex object that is not of function type. Same as $\mathsf{IsEqType}\ M$ *orelse* $\mathsf{IsRecord}\ M$ *orelse* $\mathsf{IsVariant}\ M$.

- $\mathsf{IsFun}\ M$

  True iff M is a complex object of function type.

- $\mathsf{Sri}\ i\ e\ M$

  Structural recursion on the insert presentation of collection types. M can be a set, a bag, or a list. Assume these identities on sets (and analogous ones on bags and lists) hold:

$$\begin{array}{rcl}
\mathsf{Sri}\ i\ e\ \mathsf{EmptySet} & = & e \\
\mathsf{Sri}\ i\ e\ (\mathsf{SngSet}\ x) & = & i\ x\ e \\
\mathsf{Sri}\ i\ e\ (\mathsf{UnionSet}(X, Y)) & = & \mathsf{Sri}\ i\ (\mathsf{Sri}\ i\ e\ Y)\ X \\
\mathsf{Sri}\ i\ e\ (\mathsf{lUnionSet}(F, G)) & = & \mathsf{Sri}\ i\ (\mathsf{Sri}\ i\ e\ (G\ ()))\ (F\ ())
\end{array}$$

- $\mathsf{Sru}\ u\ f\ e\ M$

  Structural recursion on the union presentation of colelction types. M can be a set, a bag, or a list. Assume these identities on sets (and analogous ones on bags and lists) hold:

$$\begin{array}{rcl}
\mathsf{Sru}\ u\ f\ e\ \mathsf{EmptySet} & = & e \\
\mathsf{Sru}\ u\ f\ e\ (\mathsf{SngSet}\ x) & = & f\ x \\
\mathsf{Sru}\ u\ f\ e\ (\mathsf{UnionSet}(X, Y)) & = & u\ (\mathsf{Sru}\ u\ f\ e\ X)\ (\mathsf{Sru}\ u\ f\ e\ Y) \\
\mathsf{Sru}\ u\ f\ e\ (\mathsf{lUnionSet}(F, G)) & = & u\ (\mathsf{Sru}\ u\ f\ e\ (F\ ()))\ (\mathsf{Sru}\ u\ f\ e\ (G\ ()))
\end{array}$$

- Listify M

  Convert a set, a bag, or a list into an ML list.

3.4 See Also

See also Record, CONum, COString, COBool, COBase, COFun, sf COBag, COList, COSet, CORecord, COVariant, and Ordering.

See [5, 6] for a discussion on the union presentation of sets, bags, and lists.

4 COBag

4.1 Synopsis

structure COBag

Routines for manipulating bags. These routines are mainly inspired from structural recursion and the categorical notion of a monads. Many of these routines come in various flavours: eager, lazy, parallel, or mixture of these.

4.2 Signature

```
type 'a lazy = unit -> 'a

val Trim : /{|''1|}/ -> /{|''1|}/

val Empty : /{|''1|}/
val Singleton : /''1/ -> /{|''1|}/
val Union : /{|''1|}/ -> /{|''1|}/ -> /{|''1|}/
val lUnion : /{|''1|}/ -> /{|''1|}/ -> {|''1|}/
val llUnion : /{|''1|}/ -> /{|''1|}/ -> {|''1|}/
val lllUnion : /{|''1|}/ -> /{|''1|}/ -> {|''1|}/
val Insert : /''1/ -> /{|''1|}/ -> /{|''1|}/
val InsertR : /''1/ -> /{|''1|}/ -> /{|''1|}/
val InsertN : int -> /''1/ -> /{|''1|}/ -> /{|''1|}/
val InsertNR : int -> /''1/ -> /{|''1|}/ -> /{|''1|}/

val Sri : (/''1/ -> 'a -> 'a) -> 'a -> /{|''1|}/ -> 'a
val lSri : (/''1/ -> 'a lazy -> 'a lazy) -> 'a lazy -> /{|''1|}/ -> 'a lazy
val Sru : ('a -> 'a -> 'a) -> (/''1/ -> 'a) -> 'a -> /{|''1|}/ -> 'a
val lSru : ('a lazy -> 'a lazy -> 'a) -> (/''1/ -> 'a) -> 'a -> /{|''1|}/ -> 'a
val Hom :  ('a -> 'a -> 'a) -> (/''1/ -> 'a) -> 'a -> /{|''1|}/ -> 'a

val Ext : (/''1/ -> /{|''2|}/) -> /{|''1|}/ -> /{|''2|}/
val lExt : (/''1/ -> /{|''2|}/) -> /{|''1|}/ -> /{|''2|}/
val llExt : (/''1/ -> /{|''2|}/) -> /{|''1|}/ -> /{|''2|}/
val lllExt : (/''1/ -> /{|''2|}/) -> /{|''1|}/ -> /{|''2|}/

val Map : (/''1/ -> /''2/) -> /{|''1|}/ -> /{|''2|}/
val lMap : (/''1/ -> /''2/) -> /{|''1|}/ -> /{|''2|}/
val llMap : (/''1/ -> /''2/) -> /{|''1|}/ -> /{|''2|}/
val lllMap : (/''1/ -> /''2/) -> /{|''1|}/ -> /{|''2|}/

val Mk : /''1/ list -> /{|''1|}/
val MkCOSET : COSET.set -> /{|?|}/
val MkCOBAG : UnsafeCOBAG.set -> /{|?|}/

val Km : /{|''1|}/ -> /''1/ list
```

```
val KmStd : /{|''1|}/ -> /''1/ list
val KmCOBAG : /{|''1|}/ -> UnsafeCOBAG.set
val KmCOSET : /{|''1|}/ -> COSET.set
val StdId : /{|''1|}/ -> /{|''1|}/

val IsEmpty : /{|''1|}/ -> bool
val Head : /{|''1|}/ -> /''1/
val Tail : /{|''1|}/ -> /{|''1|}/
val FirstN : int -> /{|''1|}/ -> /{|''1|}/

val StopIrsOnAcc : ('a -> bool) -> (/''1/ -> 'a -> 'a) -> 'a -> /{|''1|}/ -> 'a

val Thereis : (/''1/ -> bool) -> /{|''1|}/ -> bool
val Forall : (/''1/ -> bool) -> /{|''1|}/ -> bool

val Split : (/''1/ -> bool) -> /{|''1|}/ -> /{|''1|}/ * /{|''1|}/
val Filter : (/''1/ -> bool) -> /{|''1|}/ -> /{|''1|}/
val lFilter : (/''1/ -> bool) -> /{|''1|}/ -> /{|''1|}/
val FilterCount : (/''1/ -> bool) -> /{|''1|}/ -> int

val NumItems : /{|''1|}/ -> int
val Member : /''1/ -> /{|''1|}/ -> bool
val Subbag : /{|''1|}/ -> /{|''1|}/ -> bool
val Least : /{|''1|}/ -> /{|''1|}/ option
val Greatest : /{|''1|}/ -> /{|''1|} option
val Min : /{|''1|}/ -> /{|''1|}/ -> /{|''1|}/
val Max : /{|''1|}/ -> /{|''1|}/ -> /{|''1|}/
val Monus : /{|''1|}/ -> /{|''1|}/ -> /{|''1|}/
val Unique : /{|''1|}/ -> /{|''1|}/

val Index : (/''1/ -> /''2/) -> /{|''1|}/ -> (/''2/ -> /{|''1|}/) *
     (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) *
     (/''2/ -> /{|''1|}/)
```

4.3 Description

- Trim M

  The identity function on bag. However, it has the following side effect: Kleisli uses the union presentation of bags. That is, a bag is representated as a "union" of singleton bags and empty bags. Trim M removes leading empty bags from M.

- Empty

  The empty bag.

- Singleton M

  The singleton bag containing M.

- Union N M

The additive union of bags N and M. For example, if N is a bag having 2 apples and 1 orange and M is a bag having 3 apples and 2 watermelons, then Union N M is a bag having 5 apples, 2 watermelons, and 1 orange.

- lUnion N M

  The lazy version of Union.

- llUnion N M

  The parallel version of Union.

- lllUnion N M

  The parallel version of lUnion.

- Insert N M

  Same as Union (Singleton N)M.

- InsertR N M

  Same as Insert. However, N is positioned at the back of M.

- InsertN n N M

  The bag formed by adding n copies of N to the bag M. Assume n is positive.

- InsertNR n N M

  Same as InsertN. However, the n copies of N are positioned at the back of M.

- Sri i e M

  Structural recursion on the insert presentation of M. Assume these identities:

$$\begin{aligned} \mathsf{Sri}\ i\ e\ \mathsf{Empty} &= e \\ \mathsf{Sri}\ i\ e\ (\mathsf{Insert}\ x\ X) &= i\ x\ (\mathsf{Sri}\ i\ e\ X) \end{aligned}$$

- lSri i e M

  The lazy version of Sri.

- Sru u f e M

  Structural recursion on the union presentation of bag. Assume these identities:

$$\begin{aligned} \mathsf{Sru}\ u\ f\ e\ \mathsf{Empty} &= e \\ \mathsf{Sru}\ u\ f\ e\ (\mathsf{Singleton}\ x) &= f\ x \\ \mathsf{Sru}\ u\ f\ e\ (\mathsf{Union}\ X\ Y) &= u\ (\mathsf{Sru}\ u\ f\ e\ X)\ (\mathsf{Sru}\ u\ f\ e\ Y) \end{aligned}$$

- lSru u f e M

  The lazy version of Sru.

- Hom u f e M

  Same as Sru. However, duplicates are removed from M first.

- Ext f M

  The bag obtained by applying f to every element of M and then taking their union. That is, Ext f M $=$ Sru Union f Empty M. In fact,

- lExt f M

  The lazy version of Ext. That is, lExt f M $=$ lSru lUnion f Empty M.

- llExt f M

  The parallel version of Ext. The current implementation of this operator is given below.

```
fun llExt F N =
   let val Ch = channel()
       val No = Sru
            (SupportB.Curry op+)
            (fn X => 1 before spawn (fn() => send (Ch, F X)))
            0 N
   in Iterate.iterate (fn X => Union X (accept Ch)) No Empty
   end
```

- lllExt f M

  The parallel version of lExt. The implementation of this operator is more complicated and is given below. The use of Trim in the implementation forces lllExt to always compute a first element if it exists. It can be removed to achieve a greater degree of laziness.

```
fun lllExt F N =
   let
      (* Channel for items and outermost structure. *)
      val Ch = channel ()
      val Ch' = channel ()

      (* Apply F to elem and send result to channel.
       * New channel is created to keep intermediate structure. *)
      fun ApplyF X Ch' =
         let val Ch'' = channel ()
         in spawn (fn() => send (Ch, F X));
            spawn (fn () => send (Ch',
               lUnion (fn() => accept Ch) (fn() => accept Ch'')));
            Ch''
         end
   in spawn (fn() => send (Sri ApplyF Ch' N, Empty));
      Trim (accept Ch')
   end
```

- Map f M

  The bag obtained by applying f to every element in M. Hence Map f M $=$ Ext (Singleton$\circ$ f) M.

- lMap f M

  The lazy version of Map.

- ||Map f M

  The parallel version of Map.

- |||Map f M

  The parallel version of ||Map.

- Mk N

  The bag formed by the elements on the ML list N.

- MkCOSET

  ???

- MkCOBAG

  ???

- Km M

  The ML list formed by enumerating the elements in M. The ordering is random. The number of duplicates are preserved.

- KmCOBAG M

  The inverse of MkCOBAG.

- KmCOSET M

  The inverse of MkCOSET.

- KmStd M

  The ML list formed by enumerating the elements of M. The list is ascending. The number of duplicates are preserved.

- StdId M

  The identity function on bag. However, it actually sorts the bag ascendingly. Naturally number of duplicates is preserved.

- IsEmpty M

  True iff the bag M is empty.

- Head M

  A random element in M. Assume M is not empty. The effect is same as $hd(\mathsf{Km}\ M)$.

- Tail M

  The bag formed by removing one copy of Head$H$.

- FirstN n M

  Return upto the first n elements in M.

- StopsIrsOnAcc $p\ i\ e\ M$

  Apply i to elements of M using Sri until p is true. Hence, StopIrsOnAcc $p\ i\ e\ M\ =\ \text{Sri}\ f\ e\ M$, where $f\ x\ a\ =\ a$ if $p\ a$ is true and $f\ x\ a\ =\ i\ x\ a$ otherwise.

- Thereis $p\ M$

  True iff there is an element in M making p true. Stop checking as soon as such an element is found.

- Forall $p\ M$

  True iff every element in M makes p true. Stop checking as soon as an element fails p.

- Split $p\ M$

  The pair (X, Y) such that X contains all those elements of M satisfying p and Y those elements failing p. Duplicates are preserved.

- Filter $p\ M$

  Those elements of M satisfying p.

- lFilter $p\ M$

  The lazy version of Filter.

- FilterCount $p\ M$

  The number of elements in M satisfying p.

- NumItems M

  The cardinality of M.

- Member $N\ M$

  True iff N is in M.

- Least M

  The smallest element in M.

- Greatest M

  The largest element in M.

- Subbag $N\ M$

  True iff N is a subbag of M. That is, for any element in N, the number of times it appears in N is no more than the number of times it appears in M.

- Min $N\ M$

  The “intersection” of N and M. Return a bag so that the number of times an element appears in it is equal to the minimum of the number of times it appears in N and M. For example, if N has 2 apples and 1 orange and M has 3 apples and 2 watermelons, then Min $N\ M$ has 2 apples only.

- Max N M

  Another kind of intersection of bags. Return a bag so that the number of times an element appears in it is equal to the maximum of the number of times it appears in N and M. For example, if N has 2 apples and 1 orange and M has 3 apples and 2 watermelons, then Max N M has 3 apples, 1 orange, and 2 watermelons.

- Monus N M

  The bag formed by subtracting M from N. For example, if N has 2 apples and 1 orange and M has 3 apples and 2 watermelons, then Monus N M has 1 orange.

- Unique M

  The bag formed by removing all duplicates from M.

- Index I M

  Build an index on M using the indexing function I. Return the tuple (lt, leq, eq, geq, gt), where $lt(i)$ gives all the elements m in M such that $I(m) < i$, $leq(i)$ gives all the elements m in M such that $I(m) \leq i$, $eq(i)$ gives all the elements m in M such that $I(m) = i$, $geq(i)$ gives all the elements m in M such that $I(m) \geq i$, and $gt(i)$ gives all the elements m in M such that $I(m) > i$.

4.4 See Also

See also CO, COSet, and COList.

See [6] for an introduction to structural recursion as a query paradigm. See [12, 1, 8] for a description of standard bag operators. See [13] for more information on the theory of bags. See [7] for an introduction to lazy evaluation.

5 COBase

5.1 Synopsis

structure COBase

Routines for managing extensible base types. The user must supply a linear order for each base types added to the system. He must also supply conversion routines.

5.2 Signature

```
type tno = int

val DefaultMk : tno * CO.CO -> CO.CO
val DefaultKm : tno * CO.CO -> CO.CO

val Add : string -> (CO.CO * CO.CO -> LibBase.relation) option
      -> (tno * CO.CO -> CO.CO) option -> (tno * CO.CO -> CO.CO) option -> tno

val Add' : string ->  (CO.CO * CO.CO -> LibBase.relation) option
      -> (tno * CO.CO -> CO.CO) option -> (tno * CO.CO -> CO.CO) option
      -> TY.TY list -> tno

val GetParams : tno -> TY.TY list
val GetName : tno -> string
val GetCmp : tno -> CO.CO * CO.CO -> LibBase.relation
val GetMk : tno -> tno * CO.CO -> CO.CO
val GetKm : tno -> tno * CO.CO -> CO.CO
val PeekTNo : string -> tno option

val Mk : tno -> CO.CO -> CO.CO
val Mk' : tno -> CO.CO -> CO.CO
val Km : CO.CO -> CO.CO
val Km' : CO.CO -> tno * CO.CO
```

5.3 Description

Every new base type has an internal representation and an external representations. Both representations are chosen by the user. The internal representation can be anything. But the external representation must be a normal complex object of type CO.CO. An object of a new base type is always moved in and out of the system using its external representation. But once it is inside the system, it is always kept using the internal representation. Kleisli does conversions automatically.

- DefaultMk (T, M)

Convert object M of base type T from external to internal representation. It is to be used only when the user has not specified a conversion routine for base type T.

- DefaultKm (T, M)

  Convert object M of base type T from internal to external represenation. It is to be used only when the user has not specified a conversion routine for base type T.

- Add $N\ C\ M\ K$

  Create a new base type. Call it N. Order its objects using the comparison function C. Use M to convert the external representation of this base type to its internal representation. Use K to convert the internal representation of this base type to its external representation. Return an integer, which is the number of the base type thus created. Set C, M, or K to NONE if no special ordering or conversions is required.

- Add' $N\ C\ M\ K\ P$

  Create a new parameterized base type. Call it N with parameters P, usually P is just a list of type variables. Order its objects using the comparison function C. Use M to convert the external representation of this base type to its internal representation. Use K to convert the internal representation of this base type to its external representation. Return an integer, which is the number of the base type thus created. Set C, M, or K to NONE if no special ordering or conversions is required.

- GetParams T

  Assume T is the number of a user-defined base type created by Add'. Return its parameters.

- GetName T

  Assume T is the number of a user-defined base type. Return its name.

- GetCmp T

  Assume T is the number of a user-defined base type. Return its associated linear order.

- GetMk T

  Assume T is the number of a user-defined base type. Return the function to be used to convert the external representation of this base type to its internal representation.

- GetKm T

  Assume T is the number of a user-defined base type. Return the function to be used to convert the internal representation of this base type to its external representation.

- PeekTNo S

  Check if S is the name of a user-defined base type. Return NONE if it is not. Return the base type number if it is.

- Mk $T\ M$

  Assume T is a user-defined base type. Assume M is an external representation of an object of type T. Return its internal representation.

- Mk' T M

  Assume M is object of type T in its external representation. Tag M with T.

- Km M

  Assume M is a user-defined base type object in its internal representation. Return its external representation.

- Km' M

  Assume M is a user-defined base type object in its internal representation. Return (T, N), where N is the external representation of M and T is the user-defined type of M.

5.4 See Also

See also CO.

6 COBool

6.1 Synopsis

structure COBool

Routines for manipulating Boolean objects.

6.2 Signature

```
val Mk         : bool -> /bool/
val Km         : /bool/ -> bool

val IfThenElse : /bool/ -> (unit -> '1) -> (unit -> '1) -> '1
```

6.3 Description

- Mk M

  Return a CPL Boolean object corresponding to the ML Boolean value M.

- Km M

  The inverse of Mk.

- IfThenElse M f g

  Return the value of $f()$ if M is true. Return the value of $g()$ if M is false.

6.4 See Also

See also CO.

7 COCONFIG

7.1 Synopsis

structure COCONFIG

This structure puts various control flags used in the complex object interface of Kleisli together.

7.2 Signature

```
val DefaultCmp: (CO.CO * CO.CO -> LibBase.relation) ref

val Format : FORMAT.format ref

val InitWriter :  (unit -> FORMAT.Format * (string -> unit) *
      (unit -> unit) * (unit -> unit) * string) ref

val Writer : (FORMAT.Format * (string -> unit) * (unit -> unit) *
      (unit -> unit) * string) ref

val UseRuntimeOrder : bool ref
```

7.3 Description

- DefaultCmp

  The default comparison function complex objects. Alias for COBase.DefaultCmp.

- Format

  The default format for printing complex objects. Alias for COO.Format.

- Writer

  The default writer for printing complex objects. It is advisable not to update Writer directly. To change the default writer, change InitWriter instead. Alias for COO.Writer.

- InitWriter

  The default function for changing the default writer for printing complex objects. Called automatically from Control when the CPL interpretor is invoked. Alias for COO.Init.

- UseRuntimeOrder

  Some types, like records, can be ordered in a way that is independent of the runtime state or in a way that is dependent on the runtime state. Usually the latter is more efficient to compute. So use this flag to indicate which one is preferred. Default for this flag is *true*. Alias for Ordering.UseRuntimeOrder.

7.4 See Also

See also Control, Ordering, FORMAT, COO, COBase.

8 COFun

8.1 Synopsis

structure COFun

Routines to manipulate CPL function objects.

8.2 Signature

```
infix O
infix |>

val Mk      : ('1 -> '2) -> /'1 -> '2/
val Km      : /'1 -> '2/ -> '1 -> '2

val Apply   : /'1 -> '2/ -> '1 -> '2
val Compose : /'1 -> '2/ -> /'3 -> '1/ -> '3 -> '2
val O       :  /'1 -> '2/ * /'3 -> '1/ -> '3 -> '2
val |>      : /'3 -> '1/ * /'1 -> '2/ -> '3 -> '2
```

8.3 Description

- Mk f

  Turn a ML function into a CPL object.

- Km M

  The inverse of Mk.

- Apply F M

  Apply the function F to M.

- Compose G H

  Compose functions G and H. Equivalent to (Km G) $\circ$ (Km H).

- G O H

  Same as Compose G H.

- $G \mid > H$

  Same as H O G. Function composition in "diagram" order.

8.4 See Also

See also CO.

9 COI

9.1 Synopsis

structure COI

Routines to input complex object laid out in the CPL Standard Exchange Format. Work for both compressed and uncompressed versions.

9.2 Signature

```
type parser

val OpenParserOnStream : CIO.instream -> parser
val OpenParserOnString : string -> parser
val OpenParserOnFile   : string -> parser
val OpenParserOnBigFile: string -> parser
val ReadLazy           : CO.CO CML.chan -> unit CML.chan -> parser -> unit
val ReadEager          : parser -> CO.CO
val ReadBig            : parser -> CO.CO
val EOF                : parser -> bool
val CloseParser        : parser -> unit
val ReadFromStream     : CIO.instream -> CO.CO
val ReadFromString     : string -> CO.CO
val ReadFromFile       : string -> CO.CO
val ReadFromBigFile    : string -> CO.CO
val ReadFromFileLazy   : CO.CO CML.chan -> unit CML.chan -> string -> unit
val ReadFromStreamLazy : CO.CO CML.chan -> unit CML.chan -> CIO.instream -> unit
val BlockSize          : int ref
```

9.3 Description

All input routines in this module take in a parser. Parsers are produced by the open routines in this module. These parsers expect the input data to be laid out according to the CPL Standard Exchange Format.

There are three input behaviours. “Eager” is mainly for small files and returns nothing until the whole file is read. “Lazy” is for bigger files; it returns data lazily but uses a read-ahead semantics. That is, “lazy” will return data as soon as some is available, but while the previous block of data is being process, it will go ahead to get the next block. “Big” is for reading very big files; it returns data lazily and uses a on-demand semantics.

Thus “eager” needs enough space to store the whole file, “lazy” may need enough space to store the whole file if the data it returned is not processed fast enough, and “big” needs only space for one block of data at any one time. Note that both “eager” and “lazy” hold the file descriptor for the entire duration of the read process. On the other hand, “big” holds the file descriptor for the duration of one block at any one time.

- OpenParserOnStream S

  Create a parser for the input stream S.

- OpenParserOnString S

  Create a parser for the string S.

- OpenParserOnFile F

  Create a parser for the input file F.

- OpenParserOnBigfile F

  Create a parser for the big input file F.

- ReadLazy O R P

  Read one complex object from the parser P. Pass the object read to channel O. Signal ready on the channel R after the object has been completely read in. The object is read lazily. That is, if it is a set, it will be available at the other end of O as soon as its first element is parsed. The system will fill in subsequent elements automatically later on — actually the system will fill in these elements while you are busy processing the earlier ones!

- ReadEager P

  Read one complex object from parser P. Return the object read.

- ReadBig P

  Read one complex object from parser P. The object is read lazily. That is, if it is a set, as soon as its first element is parsed, the set will be made available. The system will fill in subsequent elements automatically later on, when these elements are needed.

- EOF P

  True iff there is no more object available on P.

- CloseParser P

  Close the parser P. No more object can be read from it.

- ReadFromStream S

  Read the complex object in the stream S.

- ReadFromString S

  Read the complex object in the string S.

- ReadFromFile F

  Read the complex object in the file F.

- ReadFromBigFile F

  Read the complex object in the file F. The reading is lazy. That is, if the object is a large set, it will be available as soon as the first element is read. The system will fill in subsequent elements automatically later on when there are needed.

- ReadFromFileLazy O R F

  Read the complex object in the file F. Pass the object read to channel O. Signal ready on channel R after the object is fully read. The reading is lazy. That is, if the object is a large set, it will be available on O as soon as the first element is read. The system will fill in subsequent elements automatically later on, in parallel to the processing of the earlier elements.

- ReadFromStreamLazy O R S.

  Read the complex object in the stream S. Pass the object read to channel O. Signal ready on channel R after the object is fully read. The reading is lazy. That is, if the object is a large set, it will be available on O as soon as the first element is read. The system will fill in subsequent elements automatically later on.

- BlockSize

  The block size used for reading big files. Default is 20K at a time.

9.4 See Also

See also STDIO, MachineO, CPLFORMAT and the section on "CPL Standard Exchange Format."

10 COList

10.1 Synopsis

structure COList

Routines for manipulating lists. These routines are mainly inspired from structural recursion and the categorical notion of a monads. Many of these routines come in various flavours: eager, lazy, parallel, or mixture of these.

10.2 Signature

```
type 'a lazy = unit -> 'a

val Trim      : /[''1]/ -> /[''1]/

val Empty     : /[''1]/
val Append    : /[''1]/ -> /[''1]/ -> /[''1]/
val lAppend   : /[''1]/ -> /[''1]/ -> /[''1]/
val llAppend  : /[''1]/ -> /[''1]/ -> /[''1]/
val lllAppend : /[''1]/ -> /[''1]/ -> /[''1]/
val Singleton : /''1/ -> /[''1]/
val Insert    : /''1/ -> /[''1]/ -> /[''1]/
val InsertR   : /''1/ -> /[''1]/ -> /[''1]/

val Sri  : (/''1/ -> 'a -> 'a) -> 'a -> /[''1]/ -> 'a
val lSri : (/''1/ -> 'a lazy -> 'a lazy) -> 'a lazy -> /[''1]/ -> 'a lazy
val Irs  : (/''1/ -> 'a -> 'a) -> 'a -> /[''1]/ -> 'a
val Sru  : ('a -> 'a -> 'a) -> (/''1/ -> 'a) -> 'a -> /[''1]/ -> 'a
val lSru :  ('a lazy -> 'a lazy -> 'a) -> (/''1/ -> 'a) -> 'a -> /[''1]/ -> 'a

val Ext   : (/''1/ -> /[''2]/) -> /[''1]/ -> /[''2]/
val lExt  : (/''1/ -> /[''2]/) -> /[''1]/ -> /[''2]/
val llExt : (/''1/ -> /[''2]/) -> /[''1]/ -> /[''2]/
val lllExt: (/''1/ -> /[''2]/) -> /[''1]/ -> /[''2]/

val Map       : (/''1/ -> /''2/) -> /[''1]/ -> /[''2]/
val lMap      : (/''1/ -> /''2/) -> /[''1]/ -> /[''2]/
val llMap     : (/''1/ -> /''2/) -> /[''1]/ -> /[''2]/
val lllMap    : (/''1/ -> /''2/) -> /[''1]/ -> /[''2]/

val Mk        : /''1 list -> /[''1]/
val MkCOSET   : COSET.set -> /[?]/
val Km        : /[''1]/ -> /''1/ list
val KmCOSET   : /[''1]/ -> COSET.set

val IsEmpty   : /[''1]/ -> bool
val Head      : /[''1]/ -> /''1/
```

```
val Tail      : /[’’1]/ -> /[’’1]/
val FirstN    : int -> /[’’1]/ -> /[’’1]/

val StopIrsOnAcc : (’a -> bool) -> (/’’1/ -> ’a -> ’a) -> ’a -> /[’’1]/ -> ’a
val StopSriOnAcc : (’a -> bool) -> (/’’1/ -> ’a -> ’a) -> ’a -> /[’’1]/ -> ’a

val Forall    : (/’’1/ -> bool) -> /[’’1]/ -> bool
val Thereis   : (/’’1/ -> bool) -> /[’’1]/ -> bool

val Filter    : (/’’1/ -> bool) -> /[’’1]/ -> /[’’1]/
val lFilter   : (/’’1/ -> bool) -> /[’’1]/ -> /[’’1]/
val FilterCount : (/’’1/ -> bool) -> /[’’1]/ -> int
val Split     : (/’’1/ -> bool) -> /[’’1]/ -> /[’’1]/ * /[’’1]/

val NumItems  : /[’’1]/ -> int
val Member    : /’’1/ -> /[’’1]/ -> bool
val Least     : /[’’1]/ -> /’’1/ option
val Greatest  : /[’’1]/ -> /’’1/ option
val Prefix    : int -> /[’’1]/ -> /[’’1]/
val Postfix   : int -> /[’’1]/ -> /[’’1]/
val Suffix    : int -> /[’’1]/ -> /[’’1]/
val SubList   : int -> int -> /[’’1]/ -> /[’’1]/

val Rev       : /[’’1]/ -> /[’’1]/
val Sort      : /[’’1]/ -> /[’’1]/
val SortDesc  : /[’’1]/ -> /[’’1]/
val GenSort   : (/’’1/ * /’’1/ -> bool) -> /[’’1]/ -> /[’’1]/
val UniqueSort : /[’’1]/ -> /[’’1]/
val UniqueSortDesc   : /[’’1]/ -> [’’1]/

val Index     :  (/’’1/ -> /’’2/) -> /[’’1]/
       -> (/’’2/ -> /[’’1]/) * (/’’2/ -> /[’’1]/)
          * (/’’2/ -> /[’’1]/) * (/’’2/ -> /[’’1]/) * (/’’2/ -> /[’’1]/)
```

10.3 Description

- Trim M

  The identity function on lists. However, it has the following side effect: Kleisli uses the union presentation of lists. That is, a list is representated as a concatenation of singleton lists and empty lists. Trim M removes leading empty lists from M.

- Empty

  The empty list.

- Append N M

  The list formed by appending M to the back of N.

- lAppend N M

The lazy version of Append.

- llAppend N M

  The parallel version of Append

- lllAppend N M

  The lazy version of llAppend.

- Singleton M

  The singleton list containing M.

- Insert N M

  The list formed by adding N to the front of M.

- InsertR N M

  The list formed by adding N to the back of M.

- Sri i e M

  Structural recursion on the insert presentation of lists. Assume these identities:

$$\begin{aligned} \textsf{Sri}\ i\ e\ \textsf{Empty} &= e \\ \textsf{Sri}\ i\ e\ (\textsf{Insert}\ x\ X) &= i\ x\ (\textsf{Sri}\ i\ e\ X) \end{aligned}$$

- lSri i e M

  The lazy version of Sri.

- lrs i e M

  Equivalent to Sri i e (Rev M). Hence these identities are assumed:

$$\begin{aligned} \textsf{lrs}\ i\ e\ \textsf{Empty} &= e \\ \textsf{lrs}\ i\ e\ (\textsf{Insert}\ x\ X) &= \textsf{lrs}\ i\ (i\ x\ e)\ X \end{aligned}$$

- Sru u f e M

  Structural recursion on the union presentation of lists. Assume these identities:

$$\begin{aligned} \textsf{Sru}\ u\ f\ e\ \textsf{Empty} &= e \\ \textsf{Sru}\ u\ f\ e\ (\textsf{Singleton}x) &= f\ x \\ \textsf{Sru}\ u\ f\ e\ (\textsf{Append}\ X\ Y) &= u\ (\textsf{Sru}\ u\ f\ e\ X)\ (\textsf{Sru}\ u\ f\ e\ Y) \end{aligned}$$

- lSru u f e M

  The lazy version of Sru.

- Ext f M

  Apply f to each element in M and concatenate the resulting lists together. Equivalent to Sru Append f Empty M.

- lExt f M

  The lazy version of Ext. Equivalent to lSru lAppend f Empty M.

- llExt f M

  The parallel version of Ext. The implementation of this operator is given below.

```
fun llExt F S =
   let (* Apply F to each elem in parallel.
        * A new channel is created for each application.
        * This is necessary in maintaining positional ordering. *)
      val Results = Sri (fn X => fn Y =>
               let val Ch = channel()
               in (Ch :: Y) before spawn (fn() => send (Ch, F X))
               end) [] S
   in fold (fn (Ch, Y) => Append (accept Ch) Y) Results Empty
   end
```

- lllExt f M

  The parallel version of lExt. The implementation for this operator is given below.

```
fun lllExt F S =
   let (* Channel for first item. *)
      val Ch = channel ()
      (* Apply F to elem and send result to channel.
       * Extra channels are used to maintain position order. *)
      fun ApplyF X Ch =
         let val Ch’ = channel ()
             val Ch’’ = channel ()
         in spawn (fn() => send (Ch’, F X));
            spawn (fn() => send (Ch,
               lAppend (fn() => accept Ch’) (fn () => accept Ch’’)));
            Ch’’
         end
   in spawn (fn() => send (Irs ApplyF Ch S, Empty));
      Trim (accept Ch)
   end
```

- Map f M

  Apply f to each element in M. Equivalent to Ext (Singleton *circ* f) M.

- lMap f M

  The lazy version of Map.

- llMap f M

  The parallel version of Map.

- lllMap f M

  The parallel version of lMap.

- Mk L

  Build a list from the ML list L.

- MkCOSET

  ???

- Km M

  The ML list formed by enumerating elements on M.

- KmCOSET

  ???

- IsEmpty M

  True iff M is the empty list.

- Head M

  The first element in M. Assume M is not empty.

- Tail M

  All but the first element in M.

- FirstN n M

  Return upto the first n elements in M.

- StopIrsOnAcc p i e M

  Apply i to elements of M until p is true. Traverse M from front to back. Hence, StopIrsOnAcc $p\ i\ e\ M$ $=$ Sri $f\ e\ M$, where $f\ x\ a\ =\ a$ if $p\ a$ is true and $f\ x\ a\ = i\ x\ a$ otherwise.

- StopSriOnAcc p i e M

  Apply i to elements of M until p is true. Traverse M from back to front. Hence, StopIrsOnAcc $p\ i\ e\ M$ $=$ Irs $f\ e\ M$, where $f\ x\ a\ =\ a$ if $p\ a$ is true and $f\ x\ a\ = i\ x\ a$ otherwise.

- Thereis p M

  True iff there is an element in M satisfying p. Stop checking as soon as such an element is found.

- Forall p M

  True iff every element in M satisfies p. Stop checking as soon as a falsifying element is found.

- Filter p M

  Those elements of M satisfying p.

- lFilter p M

  The lazy version of Filter.

- FilterCount p M

  The number of elements in M satisfying p.

- Split p M

  The pair (X, Y), where X are those elements of M satisfying p and Y are those elemetns of M failing p.

- NumItems M

  The length of M.

- Member N M

  True iff N is in M.

- Least M

  Smallest element in M.

- Greatest M

  Largest element in M.

- Prefix n M

  The first n elements of M.

- Suffix n M

  All but the first n elements of M.

- Postfix n M

  The last n elements of M.

- SubList n m M

  Skip the first n elements of M and return the next m elements.

- Rev M

  The list formed by reversing M.

- Sort M

  The list formed by sorting M into ascending ordering. Duplicates are not removed.

- UniqueSort M

  The list formed by sorting M in ascending order. Also eliminate duplicates.

- GenSort grt M

  Treat grt as the "greater than" predicate. Return the list formed by sorting M ascendingly with respect to grt. Duplicates are not removed.

- SortDesc M

  The list formed by sorting M into descending ordering.

- UniqueSortDesc M

  The list formed by sorting M in descending order. Also remove duplicates.

- Index I M

  Build an index on M using the indexing function I. Return the tuple (lt, leq, eq, geq, gt), where $lt(i)$ is the list of elements m in M such that $I(m) < i$, $leq(i)$ is the list of elements m in M such that $I(m) \leq i$, $eq(i)$ is the list of elements m in M such that $I(m) = i$, $geq(i)$ is the list of elements m in M such that $I(m) \geq i$, $gt(i)$ is the list of elements m in M such that $I(m) > i$.

10.4 See Also

See also CO, COBag, and COSet.

See [5] for the theory of structural recursion. See [6] for the monad view of list.

11 CONum

11.1 Synopsis

structure CONum

Routines for manipulating numbers. Numbers can be integers or reals. Conversion between them are automatic.

11.2 Signature

```
val Mk     : int -> /num/
val MkInt  : int -> /num/
val MkReal : real -> /num/

val Km     : /num/ -> int
val KmInt  : /num/ -> int
val KmReal : /num/ -> real

val Add    : /num/ -> /num/ -> /num/
val Minus  : /num/ -> /num/ -> /num/
val Mult   : /num/ -> /num/ -> /num/
val Mod    : /num/ -> /num/ -> /num/
val Div    : /num/ -> /num/ -> /num/
val DivR   : /num/ -> /num/ -> /num/
```

11.3 Description

- Mk n

  Turn the ML integer n into a numeric complex object.

- MkInt n

  Same as Mk.

- MkReal n

  Turn the ML real n into a numeric complex object.

- Km M

  Turn a numeric complex object into an ML integer. Do truncation if necessary.

- KmInt M

  Same as Km.

- KmReal M

  Turn a numeric complex object into a ML real.

- Add N M

  Add M and N.

- Minus N M

  Subtract M from N.

- Mult N M

  Multiply M and N.

- Mod N M

  The remainder of N divided by M.

- Div N M

  N divided by M. Integer division.

- DivR N M

  N divided by M. Real division.

11.4 See Also

See also CO.

12 COO

12.1 Synopsis

structure COO

Routines to output complex objects in a human-readable way. Used by the CPL interpretor to display results.

12.2 Signature

```
type 'a writer

val Format             : FORMAT.Format ref
val FlushWriter        :  'a writer -> unit
val OpenWriterOnStream : CIO.outstream -> unit writer
val OpenWriterOnFile   : string -> unit writer
val AppendWriterOnFile : string -> unit writer
val OpenWriterOnString : string -> string writer
val WriteString        : 'a writer -> string -> unit
val Write              : 'a writer -> CO.CO -> unit
val CloseWriter        : 'a writer -> 'a
val WriteToFile        : string -> CO.CO -> unit
val WriteToStream      : CIO.outstream -> CO.CO -> unit
val WriteToString      : string -> CO.CO -> string
val Init               : (unit -> unit writer) ref
val Writer             : unit writer ref
```

12.3 Description

All writing routines in this module require writers to be supplied. All opening routines in this module produce writers. This module is a simple wrapper around FCOO.

- Format

  Install here the default format that is used when writers are opened by routines in this module. The normal format installed here is CPLFORMAT.StandardCPL.

- FlushWriter W

  Flush the writer W.

- OpenWriterOnStream S

  Create a writer that writes to the output stream S.

- OpenWriterOnFile F

Create a writer that writes to the file F. Note that existing file having the same name as F will be overwritten.

- AppendWriterOnFile F

  Create a writer that appends to the file F.

- OpenWriterOnString S

  Create a writer that appends to the string S. All strings written by this writer will be accumulated and returned when the writer is closed.

- WriteString W S

  Output the string S using writer W.

- Write W M

  Output the complex object M using the writer W.

- CloseWriter W

  Close the writer W. If W was created by OpenWriterOnString, the result will be a string formed by the concatenation of all the strings written by W in chronological order.

- WriteToFile F M

  Output the complex object M to the file F.

- WriteToStream S M

  Output the complex object M to the stream S.

- WriteToString S M

  Append the complex object M to the string S.

- Init

  Install here the function that is to be used to create the default writer. The CPL interpretor calls this function to obtain its output writer.

- Writer

  Install the default writer here. If you want to change the default writer used by the CPL interpretor, chnage Init rather than this place.

12.4 See Also

See also COCONFIG, CPLFORMAT, FCOO, and CO.

13 CORecord

13.1 Synopsis

structure CORecord

Routines for manipulating complex objects of record type. Implemented on top of Label, Record, and Remy modules.

13.2 Signature

```
type labno = int (* label number *)
type rno = int (* remy directory number *)
type labstr = string (* label string *)

val Mk       : labno list -> /?/ list -> /(#?:?, ..., #?:?)/
val MkRNo    : rno -> labno list -> /?/ list -> /(#?:?, ..., #?:?)/
val MkRemy   : rno -> /?/ Array.array -> /(#?:?, ..., #?:?)/
val MkRec    : /?/ Record.record  -> /(#?:?, ..., #?:?)/
val MkZip    : (labno * /?/) list -> /(#?:?, ..., #?:?)/
val MkRaw    : labstr list -> /?/ list -> /(#?:?, ..., #?:?)/
val MkRNoRaw : rno -> labstr list -> /?/ list -> /(#?:?, ..., #?:?)/
val MkZipRaw : (labstr * /?/) list -> /(#?:?, ..., #?:?)/

val KmRec    : /(#?:?, ..., #?:?)/ -> /?/ Record.record
val KmRNo    : /(#?:?, ..., #?:?)/ -> rno * labno list * /?/ list
val Km       : /(#?:?, ..., #?:?)/ -> labno list * /?/ list
val KmRemy   : /(#?:?, ..., #?:?)/ -> rno * /?/ Array.array
val KmZip    : /(#?:?, ..., #?:?)/ -> (labno * /?/) list
val KmRNoRaw : /(#?:?, ..., #?:?)/ -> rno * labstr list * /?/ list
val KmRaw    : /(#?:?, ..., #?:?)/ -> labstr list * /?/ list
val KmZipRaw : /(#?:?, ..., #?:?)/ -> (labstr * /?/) list

val KmFrivolous : /(#?:?, ..., #?:?)/ -> labno list * /?/ list
val KmFrivolousRaw : /(#?:?, ..., #?:?)/ -> labstr list * /?/ list

val Proj     : labno -> /(#?:'1; '2)/ -> /'1/
val Offset   : labno -> /(#?:'1;'2)/ -> int
val Jump     : int -> /(#?:'1;'2)/ -> /'1/
val ProjFields : labno list -> /(#?:?, ..., #?:?)/ -> /?/ list

val GetLabs  : /(#?:?, ..., #?:?)/ -> labno list
val GetLabsRaw : /(#?:?, ..., #?:?)/ -> labstr list
val GetFrivolous : /(#?:?, ..., #?:?)/ -> labno list
val GetFrivolousRaw : /(#?:?, ..., #?:?)/ -> labstr list

val IsField : labno -> /(#?:?, ..., #?:?)/ -> bool
val IsFieldRaw : labstr -> /(#?:?, ..., #?:?)/ -> bool
```

```
val ModifyInPlace : /(#?:'1;'2)/ -> labno -> /'1/ -> /(#?:'1;'2)/
val Modify  : /(#?:'1;'2)/ -> labno -> /'1/ -> /(#?:'1;'2)/
val Map     : (/?/ -> /?/) -> /(#?:?, ..., #?:?)/ -> /(#?:?, ..., #?:?)/

val MkPair  : /'1/ -> /'2/ -> /(#1:'1, #2:'2)/
val MkTriple : /'1/ -> /'2/ -> /'3/ -> /(#1:'1, #2:'2, #3:'3)/
val MkTuple  : /?/ list -> /(#1:?, #2:?, ..., #n:?)/

val Proj1   : /(#1:'1; '2)/ -> /'1/
val Proj2   : /(#2:'1; '2)/ -> /'1/
val Proj3   : /(#3:'1; '2)/ -> /'1/
val ProjN   : labstr -> /(#?:'1; '2)/ -> /'1/
val Proj2_1 : /(#1:'1, #2:'2)/ -> /'1/
val Proj2_2 : /(#1:'1, #2:'2)/ -> /'2/
val Proj3_1 : /(#1:'1, #2:'2, #3:'3)/ -> /'1/
val Proj3_2 : /(#1:'1, #2:'2, #3:'3)/ -> /'2/
val Proj3_3 : /(#1:'1, #2:'2, #3:'3)/ -> /'3/

val KmPair  : /(#1:'1, #2:'2)/ -> /'1/ * /'2/
val KmTriple : /(#1:'1, #2:'2, #3:'3)/ -> '1/ * /'2/ * /'3/
val KmTuple : /(#1:?, #2:?, ..., #n:?)/ -> /?/ list
```

13.3 Description

- Mk $[L_1, \ldots, L_n]$ $[M_1, \ldots, M_n]$

  The record having fields L_1, ..., L_n. The values at these fields are M_1, ..., M_n respectively.

- MkRNo r $[L_1, \ldots, L_n]$ $[M_1, \ldots, M_n]$

  The record having fields L_1, ..., L_n. The values at these fields are M_1, ..., M_n respectively. Assume r is the Remy directory number corresponding to fields L_1, ..., L_n. More efficient than Mk.

- MkRemy r A

  Turn the ML array A into a record whose Remy directory number is r. This is a very fast low-level operation, do not use unless you know exactly what you are doing. See [25] for more information.

- MkRec R

  Turn a Record.record into a complex object of record type.

- MkZip $[(L_1, M_1), \ldots, (L_n, M_n)]$

  The record having fields L_1, ..., L_n. The values at these fields are M_1, ..., M_n respectively.

- MkRaw $[L_1, \ldots, L_n]$ $[M_1, \ldots, M_n]$

  Same as Mk. However, L_i are label strings.

- MkRNoRaw r $[L_1, \ldots, L_n]$ $[M_1, \ldots, M_n]$

  Same as MkRNo. However, L_i are label strings.

- MkZipRaw $[(L_1, M_1), \ldots, (L_n, M_n)]$

  Same as MkZip. However, L_i are label strings.

- KmRNo M

  Inverse of MkRNo.

- KmRec M

  Inverse of MkRec.

- Km M

  Inverse of Mk.

- KmRemy M

  Inverse of MkRemy.

- KmZip M

  Inverse of MkZip.

- KmRNoRaw M

  Inverse of MkRNoRaw.

- KmRaw M

  Inverse of MkRaw.

- KmZipRaw M

  Inverse of MkZipRaw.

- KmFrivolous M

  Same as Km. However, the fields are listed in "frivolous" order. The frivolous order normally corresponds to the order in which the fields are given when the record was created.

- KmFrivolousRaw M

  Same as KmFrivolous. However, label strings are returned instead of label number numbers.

- Proj L M

  The value at the L field of M. Assume M has the L field.

- Offset L M

  The offset of the L field of M. That is, the position of the L field in M.

- Jump n M

  The value at position n in M. Assume n is the result of a Offset L operation on a record having the same type as M. At least twice as fast as Proj L M.

- ProjFields $[L_1, \ldots, L_n]$ M

  Return $[O_1, \ldots, O_n]$, where O_i is the value at the L_i field of M. Assume M has all the L_1, ..., L_n fields.

- GetLabs M

  The labels of M.

- GetLabsRaw M

  Same as GetLabs. However, label strings are returned instead of label numbers.

- GetFrivolous M

  Same as GetLabs. However, list labels in frivolus order.

- GetFrivolousRaw M

  Same as GetLabsRaw. However, list labels in frivolus order.

- IsField L M

  True iff L is a field of M

- IsFieldRaw L M

  Same as IsField. However, L is a label string.

- ModifyInPlace M L N

  Update the L field of M by N in place. Return the updated record.

- Modify M L N

  Make a copy of M. Change the L field of the copy yo N. Return the changed copy.

- Map f M

  Make a copy of M. Apply f to every field of the copy. Return the copy.

- MkPair N M

  The pair whose #1 field is N and whose #2 field is M.

- MkTriple N M O

  The triple whose #1 field is N, whose #2 field is M, and whose #3 field is O.

- MkTuple $[M_1, \ldots, M_n]$

  The tuple whose $\#i$ field is M_i.

- Proj1 M

  The value at the #1 field of M. Assume M has a #1 field.

- Proj2 M

  The value at the #2 field of M. Assume M has a #2 field.

- Proj3 M

  The value at the #3 field of M. Assume M has a #3 field.

- ProjN L M

  Same as Proj.

- Proj2_1 M

  The value at the #1 field of M. Assume M is a pair, ie. produced by MkPair. More efficient than Proj1.

- Proj2_2 M

  The value at the #2 field of M. Assume M is a pair, ie. produced by MkPair. More efficient than Proj2.

- Proj3_1 M

  The value at the #1 field of M. Assume M is a triple, ie. produced by MkTriple. More efficient than Proj1.

- Proj3_2 M

  The value at the #2 field of M. Assume M is a triple, ie. produced by MkTriple. More efficient than Proj2.

- Proj3_3 M

  The value at the #3 field of M. Assume M is a triple, ie. produced by MkTriple. More efficient than Proj3.

- KmPair M

  Inverse of MkPair.

- KmTriple M

  Inverse of MkTriple.

- KmTuple M

  Inverse of MkTuple.

13.4 See Also

See also CO, Label, Remy, Record, and COVariant.

See [21, 25] for more information on the use of Remy directories to provide fast field selection for polymorphic records.

14 COSet

14.1 Synopsis

structure `COSet`.

Routines for manipulating sets. These routines are mainly inspired from structural recursion and the categorical notion of a monads. Many of these routines come in various flavours: eager, lazy, parallel, or mixture of these.

14.2 Signature

```
type 'a lazy = unit -> 'a

val Trim      : /{''1}/ -> /{''1}/
val Unique    : /{''1}/ -> /{''1}/

val Empty     : /{''1}/
val Singleton : /''1/ -> /{''1}/
val Insert    : /''1/ -> /{''1}/ -> /{''1}/
val InsertR   : /''1/ -> /{''1}/ -> /{''1}/
val Union     : /{''1}/ -> /{''1}/ -> /{''1}/
val lUnion    : /{''1}/ lazy -> /{''1}/ lazy -> /{''1}/
val llUnion   : /{''1}/ lazy -> /{''1}/ lazy -> /{''1}/
val lllUnion  : /{''1}/ lazy -> /{''1}/ lazy -> /{''1}/

val Sri  : (/''1/ -> 'a -> 'a) -> 'a -> /{''1}/ -> 'a
val lSri : (/''1/ -> 'a lazy -> 'a lazy) -> 'a lazy -> /{''1}/ -> 'a lazy
val Sru  : ('a -> 'a -> 'a) -> (/''1/ -> 'a) -> 'a -> /{''1}/ -> 'a
val lSru : ('a lazy -> 'a lazy -> 'a) -> (/''1/ -> 'a) -> 'a -> /{''1}/ -> 'a
val Hom  : ('a -> 'a -> 'a) -> (/''1/ -> 'a) -> 'a -> /{''1}/ -> 'a
val lHom : (/''2/ lazy -> /''2/ lazy -> /''2/) -> (/''1/ -> /''2/) -> /''2/ -> /{''1}/ -> /''2/

val Ext       : (/''1/ -> /{''2}/) -> /{''1}/ -> /{''2}/
val lExt      : (/''1/ -> /{''2}/) -> /{''1}/ -> /{''2}/
val llExt     : (/''1/ -> /{''2}/) -> /{''1}/ -> /{''2}/
val lllExt    : (/''1/ -> /{''2}/) -> /{''1}/ -> /{''2}/

val Map       : (/''1/ -> /''2/) -> /{''1}/ -> /{''2}/
val lMap      : (/''1/ -> /''2/) -> /{''1}/ -> /{''2}/
val llMap     : (/''1/ -> /''2/) -> /{''1}/ -> /{''2}/
val lllMap    : (/''1/ -> /''2/) -> /{''1}/ -> /{''2}/

val Mk        : /''1/ list -> /{''1}/
val MkCOBAG   : UnsafeCOBAG.set -> /{?}/
val MkCOSET   : COSET.set -> /{?}/
val Km        : /{''1}/ -> /''1/ list
val KmCOBAG   : /{''1}/ -> UnsafeCOBAG.set
```

```
val KmCOSET   : /{''1}/ -> COSET.set
val KmStd     : /{''1}/ -> /''1/ list
val StdId     : /{''1}/ -> /{''1}/

val IsEmpty   : /{''1}/ -> bool
val Head      : /{''1}/ -> /''1/
val Tail      : /{''1}/ -> /{''1}/
val FirstN    : int -> /{''1}/ -> /{''1}/

val StopIrsOnAcc : ('a -> bool) -> (/''1/ -> 'a -> 'a) -> 'a -> /{''1}/ -> 'a

val Thereis   : (/''1/ -> bool) -> /{''1}/ -> bool
val Forall    : (/''1/ -> bool) -> /{''1}/ -> bool

val Split     : (/''1/ -> bool) -> /{''1}/ -> /{''1}/ * /{''1}/
val Filter    : (/''1/ -> bool) -> /{''1}/ -> /{''1}/
val lFilter   : (/''1/ -> bool) -> /{''1}/ -> /{''1}/
val FilterCount : (/''1/ -> bool) -> /{''1}/ -> int
val FilterRoughCount : (/''1/ -> bool) -> /{''1}/ -> int

val RoughCount : /{''1}/ -> int
val NumItems  : /{''1}/ -> int
val Member    : /''1/ -> /{''1}/ -> bool
val Subset    : /{''1}/ -> /{''1}/ -> bool
val Least     : /{''1}/ -> /''1/ option
val Greatest  : /{''1}/ -> /''1/ option
val Intersect : /{''1}/ -> /{''1}/ -> /{''1}/
val Difference : /{''1}/ -> /{''1}/ -> /{''1}/

val LimitedPowerset : /{''1}/ -> int -> /{{''1}}/ * /{{''1}}/
val SubsetN   : /{''1}/ -> int -> /{{''1}}/
val PowersetN : /{''1}/ -> int -> /{{''1}}/

val Index     : (/''1/ -> /''2/) -> /{''1}/
       -> (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/)
          * (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/)
```

14.3 Description

- Trim M

  The identity function. However, it has the side effect of removing leading empty sets from the union presentation of sets, used in our implementation.

- Unique M

  The identity function in the set theoretic sense. However, it removes duplicates from the underlying representation of the set M.

- Empty

  The empty set.

- Singleton M

  The singleton set containing M.

- Insert N M

  The set formed by inserting N into the set M. N appears as the first element of the resulting set, followed by elements of M.

- InsertR N M

  The set formed by inserting N into the set M. N appears as the last element of the resulting set, preceded by elements of M.

- Union N M

  The union of sets N and M.

- lUnion N M

  The lazy version of Union.

- llUnion N M

  The parallel version of Union.

- lllUnion N M

  The parallel version of lUnion.

- Sri i e M

  Structural recursion on the insert presentation of sets. Assume these identities:

$$\begin{aligned}\mathsf{Sri}\ i\ e\ \mathsf{Empty} &= e\\ \mathsf{Sri}\ i\ e\ (\mathsf{Insert}\ x\ X) &= i\ x\ (\mathsf{Sri}\ i\ e\ X)\end{aligned}$$

- lSri i e M

  The lazy version of Sri.

- Sru u f e M

  Structural recursion on the union presentation of sets. Assume these identities:

$$\begin{aligned}\mathsf{Sru}\ u\ f\ e\ \mathsf{Empty} &= e\\ \mathsf{Sru}\ u\ f\ e\ (\mathsf{Singleton}\ x) &= f(x)\\ \mathsf{Sru}\ u\ f\ e\ (\mathsf{Union}\ X\ Y) &= u\ (\mathsf{Sru}\ u\ f\ e\ X)\ (\mathsf{Sru}\ u\ f\ e\ Y)\end{aligned}$$

- lSru u f e M

  The lazy version of Sru.

- Hom u f e M

  Same as Sru, but duplicates are removed from M first.

- lHom $u\ f\ e\ M$

  The lazy version of Hom.

- Ext $f\ M$

  The set obtained by first applying f to each element of M and then taking the union of the resulting sets. That is, Ext $f\ M$ = Sru Union fEmpty$M$.

- lExt $f\ M$

  The lazy version of Ext. The implementation is given below. The use of Trim forces lExt to always evaluate the first element of the result if it exists. It can be removed to achieve greater laziness.

```
fun lExt F N = Trim (lSru lUnion F Empty N)
```

- llExt $f\ M$

  The parallel version of Ext. The implementation is given below. We exploited the semantics of sets to evalute its elements in parallel and return them in the order these evaluations finish.

```
fun llExt F S =
   let val Ch = channel()
      (* Apply F to each elem in parallel. *)
       val No = Sru
            (SupportB.Curry op+)
            (fn X => 1 before spawn (fn() => send (Ch, F X)))
            0 S
   in Iterate.iterate (fn X => Union X (accept Ch)) No Empty
   end
```

- lllExt $f\ M$

  The parallel version of lExt. The implementation is given below. Note the use of Trim to force the evaluation of the first element. Note also the first element is defined to be the first one to complete, even though its evaluation may have started later than all other elements. This is okay because we are dealing with sets.

```
fun lllExt F S =
   let (* Channel for items and outermost structure. *)
      val Ch = channel ()
      val Ch' = channel ()
      (* Apply F to elem and send result to channel.
       * New channel is created to keep intermediate structure. *)
      fun ApplyF X Ch' =
         let val Ch'' = channel ()
         in spawn (fn() => send (Ch, F X));
            spawn (fn () => send (Ch',
               lUnion (fn() => accept Ch) (fn() => accept Ch'')));
            Ch''
         end
   in spawn (fn() => send (Sri ApplyF Ch' S, Empty));
      Trim (accept Ch')
   end
```

- Map f M

  The set obtained by applying f to each element of M. That is, Map f M $=$ Sru Union (Singleton$\circ$ f) Empty M.

- lMap f M

  The lazy version of Map.

- llMap f M

  The parallel version of Map.

- lllMap f M

  The parallel version of lMap.

- Mk N

  The set formed by the elements on the ML list N.

- MkCOBAG N

  ???

- MkCOSET N

  ???

- Km M

  The ML list formed by enumerating the elements of M. The order of appearance is random and there can be duplicates.

- KmCOBAG M

  The inverse of MkCOBAG.

- KmCOSET M

  The inverse of MkCOSET.

- KmStd M

  The ML list formed by enumerating the elements of M. The list is strictly ascending.

- StdId M

  The identity function on sets. However, it actually sorts the set ascendingly and eliminates duplicates.

- IsEmpty M

  True iff M is empty set.

- Head M

  A random element of M. Assume M is not empty. The effect is same as $(hd($Km $M))$.

- Tail M

  The set formed by deleting (Head M) from M. If M contains several copies of the element being deleted, then only one copy is deleted. The effect is same as (Mk(tl(Km M))).

- FirstN n M

  Return upto the first n elements of M, according to the ordering given by (Km M).

- StopIrsOnAcc p i e M

  Apply i to elements of M using Sri until p is true. Hence StopIrsOnAcc $p\ i\ e\ M\ =$ Sri $f\ e\ M$, where $f\ x\ a\ =\ a$ if $p\ a$ is true and $f\ x\ a\ =\ i\ x\ a$ otherwise.

- Thereis p M

  True iff there is an element in M satisfying p. Stop checking as soon as a satisfying element is located.

- Forall p M

  True iff every element in M satisfies the predicate p. Stop checking as soon as a falsifying element is located.

- Split p M

  The pair (X, Y) such that X contains those elements of M satisfying p and Y those elements of M failing p.

- Filter p M

  Those elements of M satisfying p.

- lFilter p M

  The lazy version of Filter.

- FilterCount p M

  The number of distinct elements in M satisfying p.

- FilterRoughCount p M

  An estimate of the number of elements in M satisfying p. Much faster than FilterCount and is accurate if M has no duplicate.

- NumItems M

  The cardinality of M.

- RoughCount M

  An estimate of the number of elements in M. Much faster than NumItems and is accurate if M has no duplicate.

- Member N M

  True iff N is an element of M.

- Subset N M

  True iff N is a subset of M.

- Least M

  The smallest element in M.

- Greatest M

  The largest element in M.

- Intersect N M

  The intersection of the sets N and M.

- Difference N M

  The set formed by removing from N those elements of M.

- SubsetN M n

  The set of all subsets of M having cardinality equal to n.

- PowersetN M n

  The set of all subsets of M having cardinality less than or equal to n.

- LimitedPowerset M n

  The pair (PowersetN M n, SubsetN M n).

- Index I M

  Build an index for M using I as the indexing function. Return the tuple (lt, leq, eq, geq, gt), where $lt(i) = \{m \mid m \in M, I(m) < i\}$, $leq(i) = \{m \mid m \in M, I(m) \leq i\}$, $eq(i) = \{m \mid m \in M, I(m) = i\}$, $geq(i) = \{m \mid m \in M, I(m) \geq i\}$, and $gt(i) = \{m \mid m \in M, I(m) > i\}$.

14.4 See Also

See also CO, COBag, and COList.

See [5] for the theory of structural recursion. See [4] for some clever examples of programming with structural recursion. See [6, 23, 27] for the monad view of sets.

15 COString

15.1 Synopsis

structure COString

Routines for manipulating strings.

15.2 Signature

```
val Concate : CO.CO -> CO.CO -> CO.CO
val Explode : CO.CO -> CO.CO list
val Implode : CO.CO list -> CO.CO
val IsPrefix : CO.CO -> CO.CO -> int -> bool
val IsSubstring : CO.CO -> int -> CO.CO -> bool
val Km : CO.CO -> string
val Length : CO.CO -> int
val Mk : string -> CO.CO
val Span : CO.CO -> int -> int -> CO.CO
val StringTrans : CO.CO -> CO.CO -> CO.CO -> CO.CO
val Substring : CO.CO -> int -> int -> CO.CO
val Suffix : CO.CO -> int -> CO.CO
val ToLower : CO.CO -> CO.CO
val ToUpper : CO.CO -> CO.CO
val Tokenize : CO.CO -> CO.CO -> int -> CO.CO list
```

15.3 Description

- Concate N M

  Concatenate two strings N and M.

- Explode N

  Turn the string N into a list of characters, in the order they appear in N.

- Implode N

  Turn a list N of characters into a string. It is the inverse of Explode.

- IsPrefix N M n

  True iff N is prefix of M after skipping n characters.

- IsSubstring N n M

  True iff M is substring of N after skipping n characters.

- Km N

  Return the string N as a ML string.

- Mk N

  Turn the ML string N into a CPL string.

- Length N

  Length of the string N.

- Span N n m

  The substring of N that spans positions n to m inclusive.

- StringTrans n m N

  Translates N by mapping n to m.

- Substring N n m

  Skips first n characters of N. Returns the next m characters.

- Suffix N n

  Skips n characters of N. Returns the rest.

- ToLower N

  Make all letters lowercase.

- ToUpper N

  Make all letters uppercase.

- Tokenize N M n

  Splits up M by breaking it at places wherever characters in N occurs. n is ignoreed for now.

16 COUnit

16.1 Synopsis

structure COUnit

Routines for manipulating the unique object of CPL type unit.

16.2 Signature

```
val Mk : /unit/
val Km : /unit/ -> unit
```

16.3 Description

- Mk

  The unique object of CPL type unit.

- Km *M*

  Return the ML value (). Assume *M* is the unique object of CPL type unit.

16.4 See Also

See also CO.

17 COVariant

17.1 Synopsis

structure COVariant

Routines for manipulating variants. Built on top of CORecord via the duality between record and variants.

17.2 Signature

```
type labno = int (* label number *)
type rno = int (* remy directory number *)
type labstr = string (* label string *)

val Mk    : labno -> /'1/ -> /<#?:'1>/
val MkRaw : labstr -> /'1/ -> /<#?:'1>/

val Km    : /<#?:'1>/ -> labno * /'1/
val KmRaw : /<#?:'1>/ -> labstring * /'1/

val Some  : /'1/ -> /<#some:'1>/
val None  : /<#none:unit>/

val Test  : labno -> /<#?:'1;'2>/ -> bool
val TestRaw : labstr -> /<#?:'1;'2>/ -> bool

val Case  : labno list -> (/?/ -> 'a) list -> /<#?:?, ..., #?:?>/ -> 'a
val CaseRNo : rno -> labno list -> (/?/ -> 'a) list -> /<#?:?, ..., #?:?>/ -> 'a
val CaseRaw : labstring list -> (/?/ -> 'a) list -> /<#?:?, ..., #?:?>/ -> 'a
val Otherwise : labno list -> (/?/ -> 'a) list -> (unit -> 'a) -> /<#?:?, ..., #?:?>/ -> 'a
val OtherwiseRNo : rno -> labno list -> (/?/ -> 'a) list -> (unit -> 'a) -> /<#?:?, ..., #?:?>/ -> 'a
val OtherwiseRaw : labstring list -> (/?/ -> 'a) list -> (unit -> 'a) -> /<#?:?, ..., #?:?>/ -> 'a
```

17.3 Description

- Mk L N

  Turn N into a variant object with tag L.

- MkRaw L N

  Same as Mk. Howveer, L is a label string.

- Km M

  Inverse of Mk.

- KmRaw M

Inverse of MkRaw.

- Some N

  Turn N into a variant object with tag Label.Some.

- None

  The variant object with tag Label.None and unit value.

- Test L M

  True iff M is a variant object with tag L.

- TestRaw L M

  Same as Test. However L is a label string.

- Case $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$

  The function g, which when applied to a variant object with tag L_i and value M_i, return $f_i(M_i)$. Assume g will never be applied to any variant object whose tag is not one of L_1, ..., L_n.

  It is implemented via the duality between variants and records as follow:

  ```
  fun Case Labs Actions =
     let val DispatchTable = Record.Mk Labs Actions
     in fn (CO.Variant (LabNo, V)) => (Record.Proj LabNo DispatchTable) V
        |  _ => Error.ReportFatal "COVariant.Case: Type error."
     end
  ```

- CaseRNo r $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$

  Same as Case. Assume r is the Remy dfirectory number corresponding to L_1, ..., L_n. More efficient than Case.

- CaseRaw $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$

  Same as Case. However, L_i are label strings.

- Otherwise $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$ M

  Same as Case. However, when applied to a variant object whose tag is not among L_1, ..., L_n, return $M()$ instead.

- OtherwiseRNo r $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$ M

  Same as Otherwise. Assume r is Remy directory number corresponding to L_1, ..., L_n. More efficient than Otherwise.

- OtherwiseRaw $[L_1, \ldots, L_n]$ $[f_1, \ldots, f_n]$ M

  Same as Otherwise. However L_1, ..., L_n are label strings.

17.4 See Also

See also CO, CORecord.

See [10] for a database perspective of variants. See [9] for a programming language perspective of variants.

18 CVT

18.1 Synopsis

structure CVT

Routines for conversion between sets, bags, and lists. Parallel, eager, and lazy versions are available.

18.2 Signature

```
val ExtSB    :  (/''1/ -> /{|''2|}/) -> /{''1}/ -> /{|''2|}/
val ExtBS    :  (/''1/ -> /{''2}/) -> /{|''1|}/ -> /{''2}/
val ExtSL    :  (/''1/ -> /[''2]/) -> /{''1}/ -> /[''2]/
val ExtLS    :  (/''1/ -> /{''2}/) -> /[''1]/ -> /{''2}/
val ExtBL    :  (/''1/ -> /[''2]/) -> /{|''1|}/ -> /[''2]/
val ExtLB    :  (/''1/ -> /{|''2|}/) -> /[''1]/ -> /{|''2|}/

val lExtSB   :  (/''1/ -> /{|''2|}/) -> /{''1}/ -> /{|''2|}/
val lExtBS   :  (/''1/ -> /{''2}/) -> /{|''1|}/ -> /{''2}/
val lExtSL   :  (/''1/ -> /[''2]/) -> /{''1}/ -> /[''2]/
val lExtLS   :  (/''1/ -> /{''2}/) -> /[''1]/ -> /{''2}/
val lExtBL   :  (/''1/ -> /[''2]/) -> /{|''1|}/ -> /[''2]/
val lExtLB   :  (/''1/ -> /{|''2|}/) -> /[''1]/ -> /{|''2|}/

val llExtSB  :  (/''1/ -> /{|''2|}/) -> /{''1}/ -> /{|''2|}/
val llExtBS  :  (/''1/ -> /{''2}/) -> /{|''1|}/ -> /{''2}/
val llExtSL  :  (/''1/ -> /[''2]/) -> /{''1}/ -> /[''2]/
val llExtLS  :  (/''1/ -> /{''2}/) -> /[''1]/ -> /{''2}/
val llExtBL  :  (/''1/ -> /[''2]/) -> /{|''1|}/ -> /[''2]/
val llExtLB  :  (/''1/ -> /{|''2|}/) -> /[''1]/ -> /{|''2|}/

val lllExtSB :  (/''1/ -> /{|''2|}/) -> /{''1}/ -> /{|''2|}/
val lllExtBS :  (/''1/ -> /{''2}/) -> /{|''1|}/ -> /{''2}/
val lllExtSL :  (/''1/ -> /[''2]/) -> /{''1}/ -> /[''2]/
val lllExtLS :  (/''1/ -> /{''2}/) -> /[''1]/ -> /{''2}/
val lllExtBL :  (/''1/ -> /[''2]/) -> /{|''1|}/ -> /[''2]/
val lllExtLB :  (/''1/ -> /{|''2|}/) -> /[''1]/ -> /{|''2|}/
```

18.3 Description

- ExtBS f M

  Convert M to a set. Apply f to each element of the resulting set. Return the union (COSet.Union) of the results.

- ExtSB f M

Convert M to a bag having no duplicate. Apply f to each element of the resulting bag. Return the additive union (COBag.Union) of the results.

- ExtSL f M

  Convert M into a list that is sorted ascendingly and has no duplicate. Apply f to each element of the resulting list. Return the concatenation (COList.Append) of the results.

- ExtLS f M

  Convert M to a set. Apply f to each element of the resulting set. Return the union of the results.

- ExtBL f M

  Convert M to a list that is sorted ascendingly, do not remove duplicates. Apply f to each element of the resulting list. Return the concatenation of the results.

- ExtLB f M

  Convert M to a bag, do not remove duplicates. Apply f to each element of the resulting bag. Return the additive union of the results.

- lExtSB f M

  The lazy version of ExtSB.

- lExtBS f M

  The lazy version of ExtBS.

- lExtLB f M

  The lazy version of ExtLB.

- lExtBL f M

  The lazy version of ExtBL.

- lExtLS f M

  The lazy version of ExtLS.

- lExtSL f M

  The lazy version of ExtSL.

- llExtSB f M

  The parallel version of ExtSB.

- llExtBS f M

  The parallel version of ExtBS.

- llExtLB f M

  The parallel version of ExtLB.

- llExtBL f M

  The parallel version of ExtBL.

- llExtLS f M

  The parallel version of ExtLS.

- llExtSL f M

  The parallel version of ExtSL.

- lllExtSB f M

  The parallel version of lExtSB.

- lllExtBS f M

  The parallel version of lExtBS.

- lllExtLB f M

  The parallel version of lExtLB.

- lllExtBL f M

  The parallel version of lExtBL.

- lllExtLS f M

  The parallel version of lExtLS.

- lllExtSL f M

  The parallel version of lExtSL.

18.4 See Also

See also COBag, COList, and COSet.

19 Label

19.1 Synopsis

structure Label

Routines for managing label strings. Basically provides one-one mapping between integers and registered strings.

19.2 Signature

```
type labno = int

val Mk      : string -> labno
val Km      : labno -> string

val One     : labno
val Two     : labno
val Three   : labno
val Numeric : int -> labno list
val Some    : labno
val None    : labno

val Cmp     : labno * labno -> LibBase.relation
val Eq      : labno -> labno -> bool
val Leq     : labno -> labno -> bool
val Lt      : labno -> labno -> bool
```

19.3 Description

- Mk m

  The label number corresponding to the string m.

- Km m

  The string corresponding to the label number m.

- One

  The label number corresponding to the string "#1".

- Two

  The label number corresponding to the string "#2".

- Three

  The label number corresponding to the string "#3".

- Numeric m

  The ML list $[k_1, \ldots, k_m]$, where k_i is the label number corresponding to the string "#*i*" for $i = 1 \ldots m$.

- Some

  The label number corresponding to the string "#*some*".

- None

  The label number corresponding to the string "#*none*".

- Cmp (m, n)

  Return LibBase.Equal if (Eq m n). Return LibBase.Less if (Lt m n). Return LibBase.Greater otherwise.

- Eq m n

  True iff m and n are the same label number.

- Lt m n

  True iff the string corresponding to m lexico-graphically strictly precedes that of n.

- Leq m n

  True iff either (Eq m n) or (Lt m n).

20 MachineO

20.1 Synopsis

structure MachineO

Routines to output complex objects in the compressed CPL Standard Exchange Format. The output is more space economical than other formats and is much more efficient to read back in later.

20.2 Signature

```
type 'a writer

val OpenWriterOnStream : CIO.outstream -> unit writer
val OpenWriterOnFile   : string -> unit writer
val OpenWriterOnString : string -> string writer
val WriteString        : 'a writer -> string -> unit
val Write              : 'a writer -> CO.CO -> unit
val CloseWriter        : 'a writer -> 'a
val WriteToFile        : string -> CO.CO -> unit
val WriteToStream      : CIO.outstream -> CO.CO -> unit
val WriteToString      : string -> CO.CO -> string
```

20.3 Description

All writing routines in this module require writers to be supplied. All opening routines in this module produce writers. All routines output in the compressed CPL Standard Exchange Format, described in the section on COI.

- OpenWriterOnStream S

  Create a writer that writes to the output stream S.

- OpenWriterOnFile F

  Create a writer that writes to the file F. Note that existing file having the same name as F will be overwritten.

- OpenWriterOnString S

  Create a writer that appends to the string S. All strings written by this writer will be accumulated and returned when the writer is closed.

- WriteString W S

  Output the string S using writer W.

- Write W M

  Output the complex object M using the writer W.

- CloseWriter W

  Close the writer W. If W was created by OpenWriterOnString, the result will be a string formed by the concatenation of all the strings written by W in chronological order.

- WriteToFile F M

  Output the complex object M to the file F.

- WriteToStream S M

  Output the complex object M to the stream S.

- WriteToString S M

  Append the complex object M to the string S.

20.4 See Also

See also COI and the section on "CPL Standard Exchange Format."

21 Ordering

21.1 Synopsis

structure Ordering

Routines for comparing complex objects. The types of complex objects are ranked ascendingly as: string, num, user, bool, set, bag, list, record, variant, unit.

21.2 Signature

```
val UseRuntimeOrder : bool ref

val Test :  ''a -> ''a -> (''a * ''a -> bool) -> LibBase.relation

val CmpUnit : (CO.CO * CO.CO) -> LibBase.relation
val CmpBool : (CO.CO * CO.CO) -> LibBase.relation
val CmpNum : (CO.CO * CO.CO) -> LibBase.relation
val CmpString : (CO.CO * CO.CO) -> LibBase.relation
val CmpUser : (CO.CO * CO.CO) -> LibBase.relation
val CmpSet : (CO.CO * CO.CO) -> LibBase.relation
val CmpBag : (CO.CO * CO.CO) -> LibBase.relation
val CmpList : (CO.CO * CO.CO) -> LibBase.relation
val CmpRecord : (CO.CO * CO.CO) -> LibBase.relation
val CmpVariant : (CO.CO * CO.CO) -> LibBase.relation

val Cmp : (CO.CO * CO.CO) -> LibBase.relation
val Eq : CO.CO -> CO.CO -> bool
val Leq : CO.CO -> CO.CO -> bool
val Lt : CO.CO -> CO.CO -> bool
val Geq : CO.CO -> CO.CO -> bool
val Gt : CO.CO -> CO.CO -> bool
val Max : CO.CO -> CO.CO -> CO.CO
val Min : CO.CO -> CO.CO -> CO.CO
val Specialize : CO.CO -> CO.CO * CO.CO -> LibBase.relation
```

21.3 Description

- UseRuntimeOrder

  Some types, like records, can be ordered in a way that is independent of the runtime state or in a way that is dependent on the runtime state. Usually the latter is more efficient to compute. So use this flag to indicate which one is preferred. Default for this flag is *true*.

- Test X Y lt

  Return LibBase.Equal if $X = Y$. Return LibBase.Less if $lt(X, Y)$. Return LibBase.Greater otherwise.

- CmpUnit (N, M)

  Compare two complex objects of unit type.

- CmpBool (N, M)

  Compare two complex objects of Boolean type.

- CmpNum (N, M)

  Compare two complex objects of numeric type.

- CmpString (N, M)

  Compare two complex objects of string type.

- CmpUser (N, M)

  Compare two complex objects of user-defined type.

- CmpSet (N, M)

  Compare two complex objects of set types.

- CmpBag (N, M)

  Compare two complex objects of bag types.

- CmpList (N, M)

  Compare two complex objects of list types.

- CmpRecord (N, M)

  Compare two complex objects of record types.

- CmpVariant (N, M)

  Compare two complex objects of variant types.

- Cmp (N, M)

  Compare two complex objects of arbitrary types. However, Cmp is less efficient than CmpSet, CmpRecord, etc.

- Eq N M

  True iff Cmp(N, M) is LibBase.Equal.

- Lt N M

  True iff Cmp(N, M) is LibBase.Less.

- Leq N M

  True iff Cmp(N, M) is not LibBase.Greater.

- Geq N M

  True iff Cmp(N, M) is not LibBase.Less.

- Gt N M

  True iff Cmp(N, M) is LibBase.Greater.

- Max N M

  The greater of N and M.

- Min N M

  The smaller of N and M.

- Specialize N

  Return a specialized operation for comparing objects having the same type as N. For example, if N is a numeric object, then Specialize N $=$ CmpNum.

21.4 See Also

See also CO.

22 Remy

22.1 Synopsis

structure Remy

Routines for one-one mapping between directory numbers and lists of label numbers (lists of field names).

22.2 Signature

```
type rno = int
type labno = Label.labno = int

val AddNew'         : labno list -> labno list -> rno
val Add             : labno list -> rno
val Mk              : labno list -> rno

val GetMagic        : rno -> int
val GetOffset       : rno -> labno -> int
val GetLabs         : rno -> labno list

val GetFrivolous    : rno -> labno list
val ChangeFrivolous : rno -> labno list -> unit

val IsField         : labno -> rno -> bool

val PairRNo         : rno
val TripleRNo       : rno
val TupleRNo        : int -> rno
```

22.3 Description

These routines provide a one-one mapping between directory numbers and lists of label numbers. Each directory is a collision-free hash table on the corresponding list of label numbers.

- AddNew' m n

  Assume that m is a sorted list of label numbers and is duplicate free. Assume that n is the "frivolous" order to be used to display the labels. Associate a new directory to m. Return the directory number.

- Add m

  Assume m is a list of label numbers. Remove duplicates from it. Sort it. Check if result is already associated with a directory. If so, return the corresponding directory number. If not, create a new directory and return its number.

- Mk m

  Same as Add.

- GetMagic m

  Return the magic number given the directory number.

- GetOffset m n

  Assume m is a directory number. Assume n is a label number of m. Return position of the field n in m.

- GetLabs m

  Return the list of label numbers associated with the directory number m.

- GetFrivolous m

  Return the list of label numbers associated with the directory number m. Arrange these numbers in the "frivolous" order.

- ChangeFrivolous m n

  Make n the new "frivolous" order of m.

- IsField m n

  Test if n is a label number of the directory number m.

- PairRNo

  The directory number of records whose fields are Label.One and Label.Two

- TripleRNo

  The directory number of records whose fields are Label.One, Label.Two, and Label.Three.

- TupleRNo m

  The directory number of records whose fields are Label.Numeric m.

22.4 See Also

See also Record and CORecord.

See [21, 25] for more information on the use of Remy directories to provide fast field selection for polymorphic records.

23 Record

23.1 Synopsis

structure Record

Polymorphic records implemented using Remy's method.

23.2 Signature

```
type rno = Remy.rno = int
type labno = Remy.labno = Label.labno = int
type 'a record = {RNo: rno, Val: 'a Array.array}

val MkRNo         : rno -> labno list -> 'a list -> 'a record
val Mk            : labno list -> 'a list -> 'a record
val MkZip         : (labno * 'a) list -> 'a record
val MkRemy        : rno -> 'a Array.array -> 'a record

val Proj          : labno -> 'a record -> 'a
val Offset        : labno -> 'a record -> int
val Jump          : int -> 'a record -> 'a
val ProjFields    : labno list -> 'a record -> 'a list

val GetLabs       : 'a record -> labno list
val GetFrivolous  : 'a record -> labno list
val IsField       : labno -> 'a record -> bool

val KmRNo         : 'a record -> rno * labno list * 'a list
val Km            : 'a record -> labno list * 'a list
val KmZip         : 'a record -> (labno * 'a) list
val KmFrivolous   : 'a record -> labno list * 'a list
val KmRemy        : 'a record -> 'a Array.array * rno

val Copy          : 'a record -> 'a record
val ModifyInPlace : 'a record -> labno -> 'a -> 'a record
val Modify        : 'a record -> labno -> 'a -> 'a record
val Map           : ('a -> 'b) -> 'a record -> 'b record

val MkPair        : 'a -> 'a -> 'a record
val Proj1         : 'a record -> 'a
val Proj2         : 'a record -> 'a
val Proj2_1       : 'a record -> 'a
val Proj2_2       : 'a record -> 'a
val KmPair        : 'a record -> 'a * 'a
val MkTuple       : 'a list -> 'a record
val KmTuple       : 'a record -> 'a list
```

23.3 Description

These routines implement Remy's proposal for poylmorphic records. Records are accessed via a collision-free hash table called the directory. Field selection has constant time characteristics under both subtype and parametric polymorphism.

- 'a record

  A Remy record is a pair *rno* and *val*. *Rno* is the directory number generated from the field labels of the record. Val is an array storing the values of fields of the record. The actual position of a field of the record in *val* is hashed using the *rno*. See [25] for detail.

- MkRNo $r\ m\ n$

  Assume m is a list of label numbers. Assume r is the directory number corresponding to m, thus $r\ =$ Remy.Mk m. Assume n is the list of values at corresponding fields in m. Return the corresponding Remy record.

- Mk $m\ n$

  The Remy record having fields m and values n at these fields.

- MkZip $[(m_1,n_1),\ldots,(m_k,n_k)]$

  The Remy record having fields m_1, ..., m_k and values n_1, ..., n_k at these fields.

- MkRemy $r\ A$

  Turn the ML array A into a record whose Remy directory number is r. This is a very fast low-level operation, do not use unless you know exactly what you are doing. See [25] for more information.

- Proj $m\ n$

  The value at the m field of record n. This works as long as n has an m field. This is also a constant time operation.

- Offset $m\ n$

  The offset of the m field of record n. Recall that the Val component of n is an array. This offet is the index to the slot of the array that keeps the m field.

- Jump $o\ n$.

  The value at the oth slot of the Val component of n. Note that Offset $m\ n\ =$ Jump Offset $m\ n)\ n)$.

- ProjFields $m\ n$

  The list of values at fields m of n.

- GetLabs n

  The labels in record n.

- GetFrivolous n

  The labels in record n, arranged in “frivolous” order. The frivolous order normally corresponds to the order in which the fields are given when the record was created.

- IsField m n

  True iff m is a field of record n.

- KmRNo n

  Inverse of MkRNo.

- Km n

  Inverse of Mk. However, fields may be rearranged.

- KmZip n

  Inverse of MkZip.

- KmRemy n

  Inverse of MkRemy.

- KmFrivolous n

  Same as Km. However, fields are returned in “frivolous” order.

- Copy n

  Make a copy of record n.

- ModifyInPlace n m v

  Change the value at the m field of record n to v. Return the modified record.

- Modify n m v

  Make a copy of record n. Change the value at the m field of the copy to v. Return the changed copy.

- Map f n

  Make a copy of record n. Apply f to the value at each field of the copy. Return the transformed copy.

- MkPair m n

  Make a pair whose first component is m and second component is n.

- KmPair n

  Inverse of MkPair.

- Proj1 n

  The value at the Label.One field of record n.

- Proj2 n

  The value at the Label.Two field of record n.

- Proj2_1 n

  Same as Proj1 n). However, n is assumed to be a pair.

- Proj2_2 n

  Same as Proj2 n) However, n is assumed to be a pair.

- MkTuple $[m_1, \ldots, m_k]$

  Create a tuple whose ith component is m_i.

- KmTuple m

  Inverse of MkTuple.

23.4 See Also

See also Remy and CORecord.

See [21, 25] for details of Remy's technique to implement polymorphic records with constant-time field selection. See [18] for a very different proposal.

Part II

Compiler Interface

24 Introduction

25 Annotations

25.1 Synopsis

structure Annotations

This module maintains many kind of annotations on external primitives and servers. Such annotations are used by the optimizer.

25.2 Signature

```
type pno = int (* primitive number *)
type sno = int (* server number *)

(* primitives for which constant-folding should not be done. *)
val AddNoFold : pno -> unit
val DelNoFold : pno -> unit
val IsNoFold  : pno ->bool

(* What primitives always examine their input completely? *)
val AddEager : pno -> unit
val DelEager : pno -> unit
val IsEager  : pno ->bool

(* a primitive P is commutative if P(x,y) = P(y,x). *)
val AddComm : pno -> unit
val DelComm : pno -> unit
val IsComm  : pno ->bool

(* a primitive P is idempotent if P(x,x) = x. *)
val AddIdem : pno -> unit
val DelIdem : pno -> unit
val IsIdem  : pno ->bool

(* A primitive P is associatve if P(x, P(y, z)) = P(P(x,y), z). *)
val AddAssoc : pno -> unit
val DelAssoc : pno -> unit
val IsAssoc  : pno ->bool

(* A monadic primitive P is idempotent if P(P(y)) = P(y). *)
val AddMIdem : pno -> unit
val DelMIdem : pno -> unit
val IsMIdem  : pno ->bool

(* What servers are SQL drivers? *)
val AddSQL : sno -> unit
val DelSQL : sno -> unit
val IsSQL  : sno ->bool
```

```
(* What primitives are available on SQL? *)
val AddSQLOp : pno -> string -> unit
val DelSQLOp : pno -> unit
val PeekSQLOp : pno -> string option

(* Is a server local? *)
val AddLocalServer : sno -> unit
val DelLocalServer : sno -> unit
val IsLocalServer  : sno ->bool

(* Is primitive cheap or should be recomputed? *)
val AddRecomputePrim : pno -> unit
val DelRecomputePrim : pno -> unit
val IsRecomputePrim  : pno -> bool

(* Does server come with its own cache? *)
val AddUseOwnCache : sno -> unit
val DelUseOwnCache : sno -> unit
val IsUseOwnCache  : sno -> bool

(* Ensure consistency when primitives are deleted *)
val DelOps         : (pno -> unit) list ref
val CallDelOps     : pno -> unit
val DelDrivers     : (sno -> unit) list ref
val CallDelDrivers : sno -> unit
```

25.3 Description

- AddNoFold P

  Indicate that P is a primitive for which constant-folding should not be done.

- IsNoFold P

  True iff P is currently AddNoFold-annotated.

- DelNoFold P

  Remove the AddNoFold-annotation on P.

- AddEager P

  Indicate that P is a primitive which examines its input completely. Hence the input to P should be evaluated eagerly.

- IsEager P

  True iff P is currently AddEager-annotated.

- DelEager P

  Remove the AddEager-annotation on P.

- AddComm P

  Indicate that P is a binary commutative primitive.

- IsComm P

  True iff P is currently AddComm-annotated.

- DelComm P

  Remove the AddComm-annotation on P.

- AddIdem P

  Indicate that P is a binary idempotent primitive.

- IsIdem P

  True iff P is currently AddIdem-annotated.

- DelIdem P

  Remove the AddIdem-annotation on P.

- AddAssoc P

  Indicate that P is a binary associative primitive.

- IsAssoc P

  True iff P is currently AddAssoc-annotated.

- DelAssoc P

  Remove the AddAssoc-annotation on P.

- AddMIdem P

  Indicate that P is a monadic idempotent primitive.

- IsMIdem P

  True iff P is currently AddMIdem-annotated.

- DelMIdem P

  Remove the AddMIdem-annotation on P.

- AddSQL S

  Indicate that S is a SQL server that accepts a restricted optimizable fragment of SQL queries.

- IsSQL S

  True iff S is currently AddSQL-annotated.

- DelSQL S

  Remove the AddSQL-annotation on S.

- AddSQLOp P S

  Indicate that primitive P is equivalent to SQL-operator S.

- DelSQLOp P

  Remove the AddSQLOp-annotation on P.

- PeekSQLOp P

  Return the SQL-operator that P implements.

- AddLocalServer S

  Indicate that server S uses only local data.

- IsLocalServer S

  True iff S is currently AddLocalServer-annotated.

- DelLocalServer S

  Remove the AddLocalServer-annotation on S.

- AddRecomputePrim P

  Indicate that primitive P is cheap or should be recomputed. For example, it might have side effects.

- DelRecomputePrim P

  Remove the AddRecomputePrim-annotation on P.

- IsRecomputePrim P

  True iff P is currently AddRecomputePrim-annotated.

- AddUseOwnCache S

  Indicate that S is a server that manages its own cache.

- DelUseOwnCache S

  Remove the AddUseOwnCache-annotation on S.

- IsUseOwnCache S

  True iff server S is AddUseOwnCache-annotated.

- DelOps

  The list of operations that remove annotations from a primitive. If you create a new kind of annotation for primitive, you should append your de-annotation procedure here.

- CallDelOps P

  Invoke all de-annotation procecures (given in DelOps) on primitive P. This is called automatically when primitive P is removed from the system.

- DelDrivers S

  The list of operations that remove annotations from a driver. If you create a new kind of annotation for drivers, you should append your de-annotation procedure here.

- CallDelDrivers S

  Invoke all de-annotation procedures (given in DelDrivers) on server S. This is called automatically when server S is removed from the system.

26 Control

26.1 Synopsis

structure Control

The control module for the CPL interpretor.

26.2 Signature

```
type parser = COI.parser
type 'a writer = 'a COO.writer

val CallCPL : WellDone.SYN -> CO.CO
val CallCPL' : string -> CO.CO
val RunCPL  : unit -> unit
val InitNonStopCPL : unit -> unit
val RunNonStopCPL : unit -> unit
val EvalLoop: parser -> unit writer -> unit
val Finish  : unit -> unit
val Init    : unit -> unit
```

26.3 Description

- EvalLoop P W

  The loop to evaluate commands returned by the parser P and writing the result out using the writer W. We present below a slightly simplified fragment of the implementation of EvalLoop to illustrate how various parts of the compiler interface are put together to execute CPL programs.

```
fun EvalLoop Parser Writer = let
   val _ = (TY.ResetVar(),             (* Reset variable counters *)
            WellDone.ResetVar(),
            MediumRare.ResetVar())
   fun EvalCommand (Rare.Quit) = true   (* Interpret commands *)
   | EvalCommand (Rare.Exec E) =
       let val MR = Rare.Cook E         (* Turn to mediumrare for type check *)
           val Ty = (case Infer.PrincipalType MR
             of SOME Ty => Ty
             | NONE => Error.ReportFatal ...)
           val WD = MediumRare.Cook MR  (* Turn to welldone for optimization *)
           val OPT = RULEDB.Optimize WD
           val CO = Interpretor.Run OPT (* execute optimized program *)
           val _ =  COO.Write Writer CO (* Display the resulting CO. *)
        in false
        end
    | ...
   in if CPLI.EOF Parser orelse EvalCommand (CPLI.Read Parser)
```

```
      then ()
      else EvalLoop Parser Writer
   end
```

- RunCPL ()

  Start a CPL interpretor session. Accept commands from keyboard. This function must not be invoked when Concurrent ML is running. Implemented as

```
fun RunCPL () = RunCML.doit (fn() =>
   let val Parser = CPLI.OpenParserOnStream std_in
   in (* Start interpreting. *)
      EvalLoop Parser (!COO.Writer);
      CPLI.CloseParser Parser
   end,
   SOME 20)
```

- RunNonStopCPL ()

  Same as RunCPL. However, this one does not stop the interpretor session on uncaught CPL error. It simply ignores the error and resume the interpretor session.

- InitNonStopCPL ()

  This function should be called before the first time you start a non-stop CPL interpretor session. It inserts the error handler used to resume the CPL interpretor session.

- CallCPL P

  Assume P is a CPL expression given in the form of a WellDone.SYN object. Run the CPL interpretor on P. Return the resulting complex object. This function must not be invoked when Concurrent ML is running. If Concurrent ML is already running and you want to evaluate P, call Interpretor.Run P instead.

- CallCPL' P

  Assume P is a CPL expression given in the form of a string. Run the CPL interpretor on P. Return the resulting complex object. This function must not be invoked when Concurrent ML is running. If Concurrent ML is already running and you want to evaluate P, call Interpretor.Run(CPLI.ReadFromStringWD P) instead.

- Init

  Call this function to register Kleisli and the CPL interpretor with Concurrent ML. This should be done once. The system can function without being registered; however, it will not be able to start most of its data drivers for remote sources.

- Finish

  Call this function to de-register Kleisli and the CPL interpretor from Concurrent ML. This function should be called to shut down the system. To restart the system, simply call Init again.

27 CMPCONFIG

27.1 Synopsis

structure CMPCONFIG

This module puts various control flags used in the compiler interface of Kleisli together.

27.2 Signature

```
val PrintRuntimeInfo             : bool ref
val ShowFinalCodes               : bool ref
val RaiseGoofedEvenWhenNotFatal  : bool ref
val TrapErr                      : bool ref
val ErrStream                    : CIO.outstream ref
val DoRun                        : bool ref
val Typed                        : bool ref
```

27.3 Description

- PrintRuntimeInfo

  Alias for Error.PrintRuntimeInfo. Set to $false$ causes Error.InfoRuntime to always discard its input.

- RaiseGoofedEvenWhenNotFatal

  Alias for Error.RaiseGoofedEvenWhenNotFatal. Set to $true$ causes Error.ReportNotFatal to always raise Error.Goofed.

- ShowFinalCodes

  Alias for Interpretor.ShowFinalCodes. Set to $true$ to display the CPL program being run. Default value is $false$. Used primarily for testing the optimizer.

- TrapErr

  Alias for Interpretor.TrapErr. Set to $true$ to trap error occurring during iterations over sets, bags, and lists., by returning empty. Default value is $false$.

- DoRun

  Alias for Interpretor.DoRun. Set to $false$ causes the interpretor to discard CPL programs submitted to it. Default value is $true$. Used primarily for testing the optimizer.

- Typed

  Alias for Infer.Typed. Set to $true$ to allow type inference. Default value is $true$.

27.4 See Also

See also Error and Interpretor.

28 CPLI

28.1 Synopsis

structure CPLI

Routines to input CPL programs and capture them as Rare.SYN, MediumRare.SYN, or WellDone.SYN objects. Rare.SYN is a representation of CPL programs that is used primarily for parsing. MediumRare.SYN is a more abstract representation used mainly for type inference. WellDone.SYN is the most abstract representation of CPL programs that is used for optimization, analysis, and compilation.

28.2 Signature

```
type parser

val OpenParserOnStream : CIO.instream -> parser
val OpenParserOnString : string -> parser
val OpenParserOnFile   : string -> parser
val Read               : parser -> Rare.SYN
val EOF                : parser -> bool
val CloseParser        : parser -> unit
val ReadFromString     : string -> Rare.SYN
val ReadFromFile       : string -> Rare.SYN
val ReadFromStream     : CIO.instream -> Rare.SYN
val ReadFromStringWD   : string -> WellDone.SYN
val ReadFromFileWD     : string -> WellDone.SYN
```

28.3 Description

All input routines in this module must be supplied with a parser. These parsers are generated by the open routines in this module.

- OpenParserOnStream S

  Create a parser to read CPL programs from stream S.

- OpenParserOnString S

  Create a parser to read CPL programs form string S.

- OpenParserOnFile F

  Create a parser to read CPL programs from file F.

- Read P

  Read the next CPL command using parser P. Return the result as a Rare.SYN object.

- EOF *P*

  True iff there is more CPL commands to be read from parser *P*.

- CloseParser *P*

  Close parser *P*.

- ReadFromString *S*

  Read the first CPL command from string *S*. Return result as a Rare.SYN object.

- ReadFromFile *F*

  Read the first CPL command from file *F*. Return result as a Rare.SYN object. Its implementation is given below.

```
fun ReadFromFile F =
   let val Parser = OpenParserOnFile F
       val T = Read Parser
       val _ = CloseParser Parser
   in T
   end
```

- ReadFromStream *S*

  Read the first CPL command from stream *S*. Return result as a Rare.SYN object.

- ReadFromStringWD *S*

  Read the first CPL command from string *S*. Return result as a WellDone.SYN object.

- ReadFromFileWD *F*

  Read the first CPL command from file *F*. Return result as a WellDone.SYN object. Its implementation is given below.

```
val ReadFromFileWD =
      let fun CVT (Rare.Exec E) = E
          | CVT _ = Error.ReportFatal
               "CPLI.ReadFromStringWD: Not a CPL expression."
      in MediumRare.Cook o Rare.Cook o CVT o ReadFromFile
      end
```

28.4 See Also

See also Rare.SYN, MediumRare.SYN, and WellDone.SYN.

For more information of CPL, consult the *Collection Programming Language Reference Manual* [24].

29 CPLO

29.1 Synopsis

structure CPLO

Routines for printing CPL expressions, captured in the form of a WellDone.SYN object, in a human-readable way.

29.2 Signature

```
val OB : string ref
val CB : string ref
val IB : string ref

val StringifyCO : CO.CO -> string

type 'a writer

val OpenWriterOnStream : CIO.outstream -> unit writer
val OpenWriterOnFile   : string -> unit writer
val AppendWriterOnFile : string -> unit writer
val OpenWriterOnString : string -> string writer
val WriteString        : 'a writer -> string -> unit
val Write              : 'a writer -> WellDone.SYN -> unit
val CloseWriter        : 'a writer -> 'a
val WriteToFile        : string -> WellDone.SYN -> unit
val WriteToStream      : CIO.outstream -> WellDone.SYN -> unit
val WriteToString      : string -> WellDone.SYN -> string
```

29.3 Description

All write routines in this module require a writer to be given. All open routines in this module produce writers.

- OB

  Brackets are used to aviod ambiguity. Set OB to the string you want to use as open bracket. Normally this is "(".

- CB

  Set CB to the string you want to use as close bracket. Normally this is ")".

- IB

  Some times ambiguity can be avoided by indentation instead of brackets. Set IB to the string you want to use as indentation. Normally this is " ".

- StringifyCO N

  Convert complex object to string. Same as COO.WriteToString "" N.

- 'a writer

  A 'a writer is a tuple ($Print$,$TidyUp$,$Indent$). $Print$ is a function to perform output. $TidyUp$ is a function to close the output buffer. $Indent$ is a string for initial indentation.

- OpenWriterOnStream S

  Create a writer that writes to stream S. Its implementation is given below.

```
fun OpenWriterOnStream Stream =
   (fn S => output (Stream, S) before flush_out Stream,  (* printer *)
    fn () => flush_out Stream,                           (* tidy up *)
    "")                                               (* indentation *)
```

- OpenWriterOnFile F

  Create a writer that writes to file F.

- AppendWriterOnFile F

  Create a writer that appends to file F.

- OpenWriterOnString S

  Create a writer that appends to string S All strings written by this writer will be accumulated and returned when the writer is closed.

- WriteString W S

  Output the string S using the writer W.

- Write W E

  Output the WellDone.SYN object E using the writer W.

- CloseWriter W

  Close the writer W. If W was created by OpenWriterOnString, return the string formed by the concatenation of all the strings written by W.

- WriteToFile F E

  Output the WellDone.SYN object E to file F.

- WriteToStream S E

  Output the WellDone.SYN object E to stream S.

- WriteToString S E

  Append the WellDone.SYN object E to string S.

29.4 See Also

See also WellDone.

30 Error

30.1 Synopsis

structure Error

Routines for printing error and runtime messages. The printing uses Concurrent ML's CIO routines.

30.2 Signature

```
exception Goofed of string

val RaiseGoofedEvenWhenNotFatal : bool ref
val PrintRuntimeInfo : bool ref
val ErrStream : CIO.outstream ref

val ReportFatal : string -> 'a
val ReportNotFatal : string -> 'a  -> 'a
val ReportRuntime  : string -> 'a
val InfoRuntime    : string -> unit
```

30.3 Description

- Goofed S

  The exception raise when error occurs within Kleisli.

- RaiseGoofedEvenWhenNotFatal

  If this flat is set to $true$, then ReportNotFatal always raises Goofed.

- PrintRuntimeInfo

  If this flag is set to $false$, then InfoRuntime always discards its input.

- ErrStream

  The output stream to send error messages to. Default is CIO.std_err.

- ReportFatal S

  Print the error message S. Raise Goofed S.

- ReportNotFatal S N

  Print the error message S. Return N. However, if RaiseGoofedEvenWhenNotFatal is set to $true$, raise Goofed S instead.

- ReportRuntime S

  Prefix S with the string "Runtime:". Print it out as an error message. Raise Goofed S.

- InfoRuntime S

  Prefix S with the string "Runtime:". Print it out on the error stream as runtime information. No error is raised.

31 Error2

31.1 Synopsis

structure Error2

Routines for printing error messages. These routines can refer to the CO structure.

31.2 Signature

```
val COWriteFn : (string -> CO.CO -> string) ref
val PrintTypeErrors : bool ref
val ReportFatalType : string -> CO.CO list -> 'a
```

31.3 Description

- COWriteFn

  Point to function that converts complex object to string. Set to COO.WriteToString by default.

- PrintTypeErrors

  Flag to control whether type errors are printed by ReportFatalType. Default is *true*.

- ReportFatalType S A

  This function is invoked when the arguments A supplied to a function do not match the input type described by the string S. If PrintTypeErrors is set, it prints out S and A. It always raises Error.Goofed.

31.4 See Also

See also COO and Error.

32 Infer

32.1 Synopsis

structure Infer

This module takes care of type inference for MediumRare.SYN objects. The type inference process is accomplished by first generating the type constraints and then unifying the resulting equations.

32.2 Signature

```
val Typed : bool ref
val PrincipalType : MediumRare.SYN -> TY.TY option
val Constraints   : MediumRare.SYN -> TY.TY -> (TY.TY * TY.TY) list
```

32.3 Description

- Typed

  Alias for CMPCONFIG.Typed. A flag to control the working of PrincipalType. It is set to *true* by default. If it set to *false* then type inference capability is turned off and PrincipalType always returns a type variable.

- Constraints E T

  Assume E is a MediumRare.SYN object. Assume T is a TY.TY object. Return a list of type equations that need to be satisfied in order for E to have type $T\theta$, where θ is any unifier of these type equations.

- PrincipalType E

  Assume E is a MediumRare.SYN object. Return its principal type. Its implementation is given below.

  ```
  fun PrincipalType E =
     let val T = TY.AnyVar (TY.FreshVar ())
         val Eqns = Constraints E T
         val Unifier = Unify.MGU Eqns
     in case Unifier of SOME mgu => SOME (mgu T) | NONE => NONE
     end
  ```

32.4 See Also

See also TY, MediumRare, Unify.

33 Interpretor

33.1 Synopsis

structure Interpretor

This module implements an interpretor for CPL. This module works on CPL expressions captured as WellDone.SYN objects.

33.2 Signature

```
val TrapErr        : bool ref
val ShowFinalCodes : bool ref
val DoRun          : bool ref
val RunSilent      : WellDone.SYN -> CO.CO
val Run            : WellDone.SYN -> CO.CO
```

33.3 Description

- TrapErr

  Set to *true* to trap error occurring during iteration over set-, bag-, and list-objects by returning a empty collection. Default value is *false*.

- ShowFinalCodes

  Set to *true* to display the CPL program being run. Default value is false. Used primary for testing the optimizer.

- DoRun

  Set to *false* causes the interpretor to discard CPL programs submitted to it. Default value is *true*. Used primary for testing the optimizer.

- Run P

  Run the CPL program P (given as a WellDone.SYN object. Return CO.Unit if the DoRun flag is false. Return the resulting complex object otherwise.

- RunSilent P

  Run the CPL program P (given as a WellDone.SYN object. Assume P is a fully eager program. Used by the optimizer to perform partial evaluation. This function is not affected by the DoRun flag.

34 Licence

34.1 Synopsis

structure Licence

This module implements a simple licence package for Kleisli.

34.2 Signature

```
val ChangeSites : string * string list -> unit
val ChangeExpiration : string * string -> unit
val Check : unit -> bool
val Version : string ref
```

34.3 Description

- ChangeSites (P, S)

  Change the allowed sites to those mentioned in S, provided P is the authentic password. If S is empty, then site restriction is disabled.

- ChangeExpiration (P, S)

  Change the expiration date to S, provided P is the authentic password. S must be a string of the form "*yyyymm*". if S is the empty string, then time restriction is disabled.

- Check

  Check whether this machine is licenced to run Kleisli. Shut everything down if machine is not licenced.

- Version

  A string storing version information.

35 MediumRare

35.1 Synopsis

structure MediumRare

The 'mediumrare' representation of CPL programs. This representation is used primarily for doing type inference.

35.2 Signature

```
type lno = int (* label number *)
type pno = int (* primitive number *)
type sno = int (* server number *)
type tno = int (* under-defined type number *)
type vno = int (* variable number *)

datatype SYN
  =  Unit | Integer of int | Real  of real          (* base types *)
  |  String of string |  Bool of bool
  |  User of tno * SYN
  |  Record of SYN Record.record                    (* records *)
  |  Proj of lno * SYN
  |  Modify of lno * SYN * SYN
  |  Variant of lno * SYN                           (* variants *)
  |  Case of SYN * (lno * TY.TY * SYN) Record.record
  |  Otherwise of SYN * (lno * TY.TY * SYN) Record.record * SYN
  |  IfThenElse of SYN * SYN * SYN
  |  Fun of vno * TY.TY * SYN | Apply of SYN * SYN  (* functions *)
  |  Primitive of pno | Variable of vno * TY.TY     (* primitives *)
  |  Error of SYN                                   (* exception *)
  |  Eq of SYN * SYN                                (* ordering *)
  |  Leq of SYN * SYN | Lt of SYN * SYN
  |  EmptySet | SngSet of SYN                       (* collections *)
  |  UnionSet of SYN * SYN
  |  ExtSet of SYN * vno * TY.TY * SYN
  |  EmptyBag |  SngBag of SYN
  |  UnionBag of SYN * SYN
  |  ExtBag of SYN * vno * TY.TY * SYN
  |  EmptyList |  SngList of SYN
  |  UnionList of SYN * SYN
  |  ExtList of SYN * vno * TY.TY * SYN
  |  ExtSB of SYN * vno * TY.TY * SYN | ExtSL of SYN * vno * TY.TY * SYN
  |  ExtLS of SYN * vno * TY.TY * SYN | ExtLB of SYN * vno * TY.TY * SYN
  |  ExtBL of SYN * vno * TY.TY * SYN | ExtBS of SYN * vno * TY.TY * SYN
  |  llExtSet of SYN * vno * TY.TY * SYN            (* parallelism *)
  |  llUnionSet of SYN * SYN
  |  llExtBag of SYN * vno * TY.TY * SYN
```

```
  |  llUnionBag of SYN * SYN
  |  llExtList of SYN * vno * TY.TY * SYN
  |  llUnionList of SYN * SYN
  |  llExtSB of SYN * vno * TY.TY * SYN | llExtBS of SYN * vno * TY.TY * SYN
  |  llExtLB of SYN * vno * TY.TY * SYN | llExtBL of SYN * vno * TY.TY * SYN
  |  llExtLS of SYN * vno * TY.TY * SYN | llExtSL of SYN * vno * TY.TY * SYN
  |  lExtSet of SYN * vno * TY.TY * SYN              (* lazy collections *)
  |  lUnionSet of SYN * SYN
  |  lExtBag of SYN * vno * TY.TY * SYN
  |  lUnionBag of SYN * SYN
  |  lExtList of SYN * vno * TY.TY * SYN
  |  lUnionList of SYN * SYN
  |  lExtSB of SYN * vno * TY.TY * SYN | lExtBS of SYN * vno * TY.TY * SYN
  |  lExtBL of SYN * vno * TY.TY * SYN | lExtLB of SYN * vno * TY.TY * SYN
  |  lExtLS of SYN * vno * TY.TY * SYN | lExtSL of SYN * vno * TY.TY * SYN
  |  lllExtSet of SYN * vno * TY.TY * SYN            (* parallel laziness *)
  |  lllUnionSet of SYN * SYN
  |  lllExtBag of SYN * vno * TY.TY * SYN
  |  lllUnionBag of SYN * SYN
  |  lllExtList of SYN * vno * TY.TY * SYN
  |  lllUnionList of SYN * SYN
  |  lllExtSB of SYN * vno * TY.TY * SYN | lllExtBS of SYN * vno * TY.TY * SYN
  |  lllExtBL of SYN * vno * TY.TY * SYN | lllExtLB of SYN * vno * TY.TY * SYN
  |  lllExtLS of SYN * vno * TY.TY * SYN | lllExtSL of SYN * vno * TY.TY * SYN
  |  Read of sno * SYN                               (* communication *)

val FreshVar : unit -> vno
val ResetVar : unit -> unit
val Cook     : SYN -> WellDone.SYN
```

35.3 Description

- SYN

  The ‘mediumrare’ representation of CPL expressions. Used primarily for type inference; hence all the TY.TY to indicate the expected type of the corresponding vno.

- FreshVar ()

  Return a fresh variable number. (Basically by incrementing an internal counter.)

- ResetVar ()

  Reset the variable counter to zero.

- Cook E

  Cook the ‘mediumrare’ representation until it becomes ‘welldone’! The ‘cooking’ is done by simply stripping off the type tags. Return the ‘welldone’ representation of the CPL program. This representation will be used by the optimizer and interpretor to compile/run the program.

35.4 See Also

See also Rare and WellDone for the other two representations of CPL programs.

36 Mutex

36.1 Synopsis

structure Mutex

This module implements a simple mutual exclusion mechanism.

36.2 Signature

```
type mutex

val MkLock : unit -> mutex
val Acquire : mutex -> unit
val Release : mutex -> unit
```

36.3 Description

- mutex

  The type of mutual exclusion lock.

- MkLock ()

  Create a new mutual exclusion lock.

- Acquire L

  Claim mutual exclusion to the lock L. Thread is suspended if L is currently possessed by another thread.

- Release L

  Relinguish mutual exclusion on the lock L. If there is a thread suspended on this lock, that thread will be resumed.

37 PChan

37.1 Synopsis

structure PChan

This module implement prioritized channels. Requests to these channels are services according to their priorities.

37.2 Signature

```
type '1a pchan
type priority = real (* priority of channel *)

val AsyncSend : '1a pchan * priority * '1a -> unit
val Send : '1a pchan * priority * '1a -> unit
val Accept : '1a pchan -> '1a
val MkChannel : unit -> '1a pchan
```

37.3 Description

- pchan

  The type of prioritized channels.

- Send (C, P, R)

  Send the request R to channel C at priority P. The send is synchronous.

- AsyncSend (C, P, R)

  Send the request R to channel C at priority P. The send is asynchronous.

- Accept C

  Accept a request of the highest priority from channel C. Thread is suspended if there is currently no request in channel C. Return the accepted request.

- MkChannel ()

  Create a new prioritized channel.

38 PQueue

38.1 Synopsis

structure PQueue

This module implements a priority queue. Priority queues are used for implementing prioritized channels.

38.2 Signature

```
type '1a pqueue
type priority = real

val MkQueue : unit -> '1a pqueue
val IsEmpty : '1a pqueue -> bool
val Enqueue : '1a pqueue * priority * '1a -> unit
val Dequeue : '1a pqueue -> priority * '1a
```

38.3 Description

- pqueue

  The type of priority queues.

- MkQueue ()

  Create a new priority queue.

- IsEmpty Q

  True iff the priority queue Q is empty.

- Enqueue (Q, P, X)

  Add an item X of priority P to the queue Q.

- Dequeue Q

  Return an item of highest priority in the queue Q.

39 Primitive

39.1 Synopsis

structure Primitive

This module provides management of extensible user-defined primitives and macros. All primitives are CO.CO objects and all macros are WellDone.SYN objects. Both primitives and macros are assigned an identifier (primitive number).

39.2 Signature

```
type pno = int (* primitive number *)

val Reset    : pno -> string -> CO.CO -> TY.TY -> unit
val Reset'   : pno -> string -> WellDone.SYN -> TY.TY -> unit
val Add      : string -> CO.CO -> TY.TY -> pno
val Add'     : string -> WellDone.SYN -> TY.TY -> unit
val Peek     : string -> pno option
val GetName  : pno -> string
val GetType  : pno -> TY.TY
val GetCO    : pno -> pno -> CO.CO
val GetMacro : pno -> WellDone.SYN
val ListAll  : unit -> string
val Delete   : pno -> unit
```

39.3 Description

- Reset P N M T

  Reset primitive/macro P. Assume P is a primitive number. Assume N is the new name for P. Assume M is the new implementation for P. Assume T is the new CPL type for P.

- Reset' P N M T

  Reset primitive/macro P. Assume P is a primitive number. Assume N is the new name for P. Assume M is the new macro definition for P. Assume T is the new CPL type for P.

- Add N M T

  Add a new primitive. Assume N is its name. Assume M is its implementation. Assume T is its type. Return the primitive number assigned to this new primitive.

- Add' N M T

  Add a new macro. Assume N is its name. Assume M is its definition. Assume T is its type. Return the primitive number assigned to this new macro.

- Peek N

  Check if N is the name of a primitive or a macro. Return the corresponding primitive number if it is. Return NONE otherwise.

- GetName P

  The name of primitive P. Assume P is a primitive number.

- GetType P

  The CPL type of P. Assume P is a primitive number.

- GetCO P

  Return the implementation of P. Assume P is a primitive number. If P is defined by as a macro, return CO.Unit.

- GetMacro P

  Return the macro definition of P. Assume P is a primitive number. If P is not defined as a macro, return WellDone.Primitive(P).

- ListAll ()

  List all the primitives and macros currently registered.

- Delete P

  Assume P is a primitive number. Remove P. Remove also all annotations about P. (Every primitive can be annotated in various ways to help the optimizer to do a better job. Annotations are implemented in another the Annotations module.)

39.4 See Also

See also Annotations for information on how to annotate primitives to help the optimizer to do a better job.

40 Rare

40.1 Synopsis

structure Rare

The ‘rare’ representation of CPL programs. This representation is very close to the concrete syntax and is used primarily by the parser.

40.2 Signature

```
datatype 'a OPT
  = INNONE
  | INS of 'a | INB of 'a | INL of 'a                  (* <-, <--, <--- *)
  | llINS of 'a | llINB of 'a | llINL of 'a         (* <<-, <<--, <<--- *)

datatype SYN
  = Unit | Integer of int | Real of real
  | String of string | Bool of bool
  | User of string * SYN
  | Record of (string * SYN) list
  | Tuple of SYN list
  | Proj of lab
  | Modify of string * SYN * SYN
  | IfThenElse of SYN * SYN * SYN          (* if SYN1 then SYN2 else SYN3 *)
  | Variant of string * SYN
  | Case of SYN * (string * SYN * SYN) list            (* SYN2 is pattern *)
  | Otherwise of SYN * (string * SYN * SYN) list * SYN (* SYN2 is pattern *)
  | Compose of SYN * SYN                                   (* SYN1 o SYN2 *)
  | Apply of SYN * SYN                                     (* SYN1 (SYN2) *)
  | Fun of SYN * SYN                                   (* SYN1 is pattern *)
  | LetIn of SYN * SYN * SYN                  (* let SYN1 == SYN2 in SYN3 *)
  | Symbol of string
  | Error of SYN
  | Eq of SYN * SYN | Leq of SYN * SYN | Lt of SYN * SYN
  | Set of SYN list | Bag of SYN list | List of SYN list
  | UnionS of SYN * SYN | UnionB of SYN * SYN | UnionL of SYN * SYN
  | ExtS of SYN * SYN OPT * SYN                        (* SYN2 is pattern *)
  | ExtB of SYN * SYN OPT * SYN                        (* SYN2 is pattern *)
  | ExtL of SYN * SYN OPT * SYN                        (* SYN2 is pattern *)
  | RSA of SYN * (SYN OPT * SYN) list                  (* SYN2 is pattern *)
  | RBA of SYN * (SYN OPT * SYN) list                  (* SYN2 is pattern *)
  | RLA of SYN * (SYN OPT * SYN) list                  (* SYN2 is pattern *)
  | lUnionS of SYN * SYN | lUnionB of SYN * SYN | lUnionL of SYN * SYN
  | lExtS of SYN * SYN OPT * SYN                       (* SYN2 is pattern *)
  | lExtB of SYN * SYN OPT * SYN                       (* SYN2 is pattern *)
  | lExtL of SYN * SYN OPT * SYN                       (* SYN2 is pattern *)
  | lRSA of SYN * (SYN OPT * SYN) list                 (* SYN2 is pattern *)
```

```
  | lRBA of SYN * (SYN OPT * SYN) list                 (* SYN2 is pattern *)
  | lRLA of SYN * (SYN OPT * SYN) list                 (* SYN2 is pattern *)
  | llUnionS of SYN * SYN | llUnionB of SYN * SYN | llUnionL of SYN * SYN
  | lllUnionS of SYN * SYN | lllUnionB of SYN * SYN | lllUnionL of SYN * SYN
  | Scanner of string * SYN
  | DontCare                                           (* _ *)
  | Slashed of string                                  (* \x *)
  | Layer of string * SYN                               (* SYN is pattern *)
  | PartialTuple of SYN list                            (* SYN is pattern *)
  | PartialRecord of (string * SYN) list                (* SYN is pattern *)

datatype COMMAND
   = Quit
   | ReadFile of string * SYN * string
   | WriteFile of SYN * SYN * string
   | Macro of string * SYN
   | Exec of SYN
   | EmbeddedSML of string

val Cook    : SYN -> MediumRare.SYN
```

40.3 Description

- 'a OPT

  The concrete syntax of CPL uses the comprehension syntax. An example comprehension is $\{e \mid pat_1$ `<-` e_1, pat_2 `<--` e_2, pat_3 `<---` e_3, e_4, pat_5 `<<-` e_5, pat_6 `<<--` e_6, pat_7 verb+¡¡—+ $e_7\}$. We use constructors of type 'a OPT to capture the body of the comprehension. In particular, INS corresponds to pat_1 `<-` e_1, INB to pat_2 `<--` e_2, INL to pat_3 `<---` e_3, INNONE to e_4, IIINS to pat_5 `<<-` e_5, IIINB to pat_6 `<<--` e_6, and IIINL to pat_7 `<<---` e_7.

- SYN

  The 'rare' representation of CPL expressions. This representation is very close to the concrete syntax and is used primarily by the parser. Hence, there are various parts that are marked by "pattern."

- COMMAND

  The representation of top-level CPL commands.

- Cook *E*

  Cook the 'rare' representation of a CPL expression until it becomes 'mediumrare'! The cooking is done by removing all sorts of patterns and syntactic sugars from the 'rare' representation and inserting type tags at appropriate places. The resulting representation will be used for type checking purposes for for further cooking into the 'welldone' representation.

40.4 See Also

See also MediumRare and WellDone.

For more information about CPL, consult the *Collection Programming Language Reference Manual* [24].

41 Server

41.1 Synopsis

structure Server

Routines for managing external data servers. Each registered server is identified by a server number. An external data server is viewed as a function to establish a connection to an external data source. A connection is viewed as a pair of functions, one for sending request to the source and one for shutting down the connection.

41.2 Signature

```
exception StartupFailed

type priority = real                      (* Priority of server access *)
type svno = int                           (* server number *)
type svfn = unit -> (dispatch * shutdown) (* server connection *)
type dispatch = CO.CO                     (* request *)
        -> CO.CO CML.chan                 (* reply to this channel *)
        -> unit CML.chan                  (* signal ready to this channel *)
        -> unit                           (* make a request to server *)
type shutdown = unit -> unit              (* shutdown server *)

val Reset : svno -> string -> svfn -> TY.TY -> (MediumRare.SYN -> TY.TY) -> int -> unit
val ResetLevel : svno -> int -> unit
val Add : string -> svfn -> TY.TY -> (MediumRare.SYN -> TY.TY) -> int -> svno
val GetName : svno -> string
val GetInputType : svno -> TY.TY
val GetOutputType : svno -> MediumRare.SYN -> TY.TY
val GetLevel : svno -> int
val GetFn : svno -> svfn
val Peek : string -> svno option
val ListAll : unit -> string list

val ResetAllConnections : svno -> unit
val ActivateAllServers  : unit -> unit
val DeactivateAllServers: unit -> unit
val Access              : svno -> priority -> CO.CO -> CO.CO
val Delete              : svno -> unit
```

41.3 Description

- StartupFailed

  This exception is raised by a data dirver to indicate that it is unable to connect to a remote data source. It is then caught by ActivateAllServers to print a non-fatal error message.

- Reset S N C T O L

  Change the external server assigned to the server number S. Assume N is the name of the new server. Assume C is the function for connecting to the new server. Assume T is the CPL type of requests to the new server. Assume O is the function for predicting the output type of the new server. Assume L is the desired level of concurrency for the new server. Kleisli will try to maintain L connections to the new server.

  The function C returns a pair (f, g). The function f takes a request, a reply channel, and a ready channel. It sends them to the server. The server sends reply to the reply channel and signals completion on the ready channel. The server assumes that synchronous communication is used on both channel. The function g is to be called when you want to shut down this server conenction.

  The function O examines the request durinmg the static type inference process at compile time; hence its input is a MediumRare.SYN object.

- ResetLevel S L

  Change the desired concurrency level of server S to L. This operation has no effect until a ResetAllConnections operation is performed on S later on.

- Add N C T O L

  Add a new external server to the system. Return the server number assigned to it. Assume N is the name of this server. Assume C is its connection function. Assume T is its request type. Assume O is the function for predicting its output type. Assume L is its desired concurrency level. See the Reset for greater detail on its parameters.

- GetName S

  The name of server S.

- GetInputType S

  The request type of server S.

- GetOutputType S

  The function for predicting the output type of server S.

- GetLevel S

  The desired concurrency level for server S.

- GetFn S

  The function for connecting to server S.

- Peek N

  Check if there is a server whose name is N. Return its server number if exists. Return NONE otherwise.

- ListAll ()

  List all registered servers.

- ResetAllConnections S

  Shut down all current connections to server S. Any pending request to S will be discarded. Open as many new ones as indicated by its desired concurrency level. Set all connections to listen for request.

- ActivateAllServers ()

  Activate all server connections.

- DeactivateAllServers ()

  Shut down all server connections.

- Access S P R

  Make a request R to server S at priority P. Return the reply from S.

- Delete S

  Remove server S from the system. Remove all annotations about S. (Servers can be annotated to help the optimizer to do a better job. Annotations are provided by the Annotations module.)

41.4 See Also

See also Annotations for how to annotate servers so that the optimizer can do a better job.

42 SyntaxTools

42.1 Synopsis

structure SyntaxTools

This module is organized into four submodules: SyntaxTool1, SyntaxTool2, SyntaxTool3, and SyntaxTool4. It provides routines for manipulation of WellDone.SYN objects. They are heavily used by the optimizer.

42.2 Signature

```
type vno = int (* variable number *)

(* Routines imported from SyntaxTool1 *)
val FV : WellDone.SYN -> IntSet.intset
val IsFree : vno -> WellDone.SYN -> bool
val IsOpen : WellDone.SYN -> bool
val IsClosed : WellDone.SYN -> bool
val Rename' : vno IntMap.intmap -> WellDone.SYN -> WellDone.SYN
val Rename : WellDone.SYN -> WellDone.SYN
val CountSub : vno ref -> WellDone.SYN -> vno -> WellDone.SYN -> WellDone.SYN
val UnsafeCountSub : vno ref -> WellDone.SYN -> vno -> WellDone.SYN -> WellDone.SYN
val Sub : WellDone.SYN -> vno -> WellDone.SYN -> WellDone.SYN
val UnsafeSub : WellDone.SYN -> vno -> WellDone.SYN -> WellDone.SYN
val Count : vno -> WellDone.SYN -> int
val FlattenTests : (WellDone.SYN -> bool)
    -> WellDone.SYN -> WellDone.SYN list * WellDone.SYN list * WellDone.SYN

(* Routines imported from SyntaxTool2 *)
val EquivFlat : WellDone.SYN -> WellDone.SYN -> bool
val Equiv : WellDone.SYN -> WellDone.SYN -> bool

(* Routines imported from SyntaxTool3 *)
val NormalizedRename : WellDone.SYN -> WellDone.SYN
val NormalizedDepth  : WellDone.SYN -> int

(* Routines imported from SyntaxTool4 *)
val Thereis : (WellDone.SYN -> bool) -> WellDone.SYN -> bool
```

42.3 Description

- FV E

  The set of free variables in E.

- IsFree x E

True iff x is a free variable in E.

- IsClosed E

  True iff E has no free variable.

- IsOpen E

  True iff E has some free variable.

- Rename' D E

  Rename all bound variables in E to fresh ones. Map free variables in E using the map D.

- Rename E

  Rename all bound variables in E to fresh ones. Preserve the free variables.

- UnsafeCountSub C E x M

  Substitute E for every x in M. Store in C the number of times x occurs in M. Assume M does not capture free variables in E Assume x is free in M.

- CountSub C E x M

  Substitute E for every x in M. Store in C the number of times x occurs in M.

- UnsafeSub E x M

  Substitute E for every x in M. Assume x is free in M. Assume M does not capture free variables of E.

- Sub E x M

  Substitute E for every x in M.

- Count x M

  The number of times x occurs free in M.

- FlattenTests Ok E

  This function flattens a tower of if-then-else tests. Assume Ok is a function to decide if a test should be extracted. Assume E is the tower of if-the-else to be flatten. Return (P, N, E'), where P is the list of tests to be 'and' together, N is the list of tests to be negated and then 'and' together. E' is the transformed unflattenable part of the tower. Satisfy: (if $\bigwedge(P) \wedge \neg(\bigvee(N))$ then E' else $\{\}) = E$.

- Equiv N M

  True if expressions N and M are equivalent modula alpha-conversion. False if do not know for sure the expressions are equal. **Caution.** N and M must not involve any lazy and/or parallel construct — those lines dealing with these constructs have been commented out as the current implementation does not need them.

- EquivFlat N M

  True iff N and M are equivalent constant expressions. The implementation is given below.

```
fun EquivFlat EmptySet EmptySet = true
| EquivFlat EmptyBag EmptyBag = true
| EquivFlat EmptyList EmptyList = true
| EquivFlat Unit Unit = true
| EquivFlat (Integer N) (Integer M) = N = M
| EquivFlat (Real N) (Real M) = N = M
| EquivFlat (Bool N) (Bool M) = N = M
| EquivFlat (String N) (String M) = N = M
| EquivFlat (Primitive P) (Primitive Q) = P = Q
| EquivFlat (Variable x) (Variable y)= x = y
| EquivFlat (Nullary N) (Nullary M) = ((Ordering.Eq N M) handle _ => false)
| EquivFlat _ _ = false
```

- NormalizedRename E

  Rename variables according to depth. Variables at the outermost level will be given the variable number 0, those at the next level will be given the number 1, and so on. For example, $\mathsf{Fun}(20, \mathsf{Fun}(4, E))$ will be renamed to $\mathsf{Fun}(0, \mathsf{Fun}(1, E))$. Essentially, the variable numbers correspond to the positions of their value on the runtime stack.

- NormalizedDepth E

  The depth of the deepest variables. This corresponds to the maximum size of the runtime stack.

- Thereis P E

  True iff there is a subexpression of E satisfying P.

43 TY

43.1 Synopsis

structure `TY`

The module defines the structure of types used by CPL. It also provides basic routines for manipulating these types. The basic type system is based on parametric polymorhism. But a limited form of (single inheritance) subtyping is supported on user-defined types thatg have no parameters.

43.2 Signature

```
type vno = int (* type variable number *)
type tno = int (* user-defined type number *)

datatype TY
      =        Unit
      |        Num
      |        String
      |        Bool
      |        User of tno * TY list
      |        Set of TY
      |        Bag of TY
      |        List of TY
      |        Fun of TY * TY    (* Fun (domain, range) *)
      |        Record of TY Record.record
      |        Variant of TY Record.record
      |        AnyVar of vno                         (* Unconstrained type*)
      |        EqVar of vno                          (* Eq type*)
      |        AnySubTy of (vno * tno)               (* Subtypes *)
                     (* AnySubTy (?, tno) can be
                      * instantiated to any subtype of tno. *)
      |        AnyRecord of vno * TY Record.record    (* Unconstrained record *)
      |        EqRecord of vno * TY Record.record     (* Eq record *)
      |        AnyVariant of vno * TY Record.record   (* Unconstrained variant*)
      |        EqVariant of vno * TY Record.record    (* Eq variant *)
      |        TupVar of int * int                   (* n-ary numeric tuple *)
                  (* TupVar (?, n) can only be instantiated
                     to k-ary numeric tuple where k >= n. *)

val NoFun       : TY -> bool
val Copy        : TY -> TY
val VarsOf      : TY -> vno list
val Occurs      : vno -> TY -> bool
val Instantiate : vno -> TY -> TY -> TY
val ResetVar    : 'a -> unit
val FreshVar    : 'a -> vno
val AddSubTy    : tno -> tno -> unit
```

```
val IsSubTyOf   : tno -> tno -> bool
```

43.3 Description

- TY

  The structure for representing CPL types.

- NoFun T

  True iff T can be turned into an eq-type. Eq-types are those that do not involve Fun.

- Copy T

  Return a copy of type T. Make all type variable fresh.

- VarsOf T

  Return all the type variables of T

- Occurs vno T

  True iff type variable vno occurs in type T. Used for implementing occurs check.

- Instantiate X T Ty

  Instantiate the type variable X to type T in type Ty. Return the instantiate type.

- ResetVar ()

  Reset the numbering of fresh variable (to zero).

- FreshVar ()

  A fresh type variable number.

- AddSubTy TNo_1 TNo_2

  Add an assertion that the user-defined type TNo_1 is a subtype of TNo_2. Assumes that TNo_1 currently has no subtypes and no supertypes.

- IsSubTyOf TNo_1 TNo_2

  True iff TNo_1 is (transitively) a subtype of TNo_2.

44 TYI

44.1 Synopsis

structure TYI

Routines to input CPL types.

44.2 Signature

```
type parser

val OpenParserOnStream : CIO.instream -> parser
val OpenParserOnString : string -> parser
val OpenParserOnFile   : string -> parser
val Read               : parser -> TY.TY
val EOF                : parser -> bool
val CloseParser        : parser -> unit
val ReadFromString     : string -> TY.TY
val ReadFromFile       : string -> TY.TY
val ReadFromStream     : CIO.instream -> TY.TY
```

44.3 Description

All input routines in this module take in a parser. Parsers are produced by the open routines in this module.

- OpenParserOnStream S

  Create a parser for the input stream S.

- OpenParserOnString S

  Create a parser for the string S.

- OpenParserOnFile F

  Create a parser for the file F.

- Read P

  Read a type using parser P.

- EOF P

  True iff there are more types to be read using parser P.

- CloseParser P

  Close parser P.

- ReadFromString S

  Read the type in the string S.

- ReadFromFile F

  Read the type in the file F.

- ReadFromStream S

  Read the type in the stream S.

44.4 See Also

See also TY and TYO.

45 TYO

45.1 Synopsis

structure TYO

Routines for printing CPL types.

45.2 Signature

```
type 'a writer

val OpenWriterOnString : string -> string writer
val OpenWriterOnStream : CIO.outstream -> unit writer
val OpenWriterOnFile   : string -> unit writer
val CloseWriter        : 'a writer -> 'a
val WriteString        : 'a writer -> TY.TY -> unit
val Write              : 'a writer -> TY.TY -> unit
val WriteToStream      : CIO.outstream -> TY.TY -> unit
val WriteToFile        : string -> TY.TY -> unit
val WriteToString      : string -> TY.TY -> string
```

45.3 Description

All output routines in this module require a writer to be supplied. All open routines in this module produce writers.

- 'a writer

  A 'a writer in this module is just a pair (*Print*,*TidyUp*). *Print* is the function to perform output. *TidyUp* is a function to close the output buffer.

- OpenWriterOnString *S*

  Create a writer that appends to the strin *S*. All strings written by this writer will be accumulated and returned when the writer is closed. Its implementation is given below.

  ```
  fun OpenWriterOnString S =
     let val Buffer = ref [S]
     in (fn S => (Buffer := (S :: (!Buffer))),
         fn _ => implode (rev (!Buffer)))
     end
  ```

- OpenWriterOnStream *S*

  Create a writer that writes to stream *S*. Its implementation is given below.

```
fun OpenWriterOnStream Stream =
   (fn S => output (Stream, S) before flush_out Stream, (* printer *)
    fn _ => ())                                          (* Tidy up *)
```

- OpenWriterOnFile S

  Create a writer that writes to file S.

- CloseWriter W

  Close the writer W. If W was created by OpenWriterOnString, the result will be a string formed by concatenating all the strings written by W.

- WriteString W S

  Write the string S using the writer W.

- Write W T

  Write the type T using the writer W.

- WriteToStream S T

  Write type T to stream S.

- WriteToFile F T

  Write type T to file F.

- WriteToString S T

  Append type T to string S.

45.4 See Also

See also TY and TYI.

46 Unify

46.1 Synopsis

structure Unify

This module implements a unification algorithm for a ML-like type system extended with polymorphic records.

46.2 Signature

```
val MGU : (TY.TY * TY.TY) list -> (TY.TY -> TY.TY) option
```

46.3 Description

- MGU E

  Assume E is a list of type equations to be unified. Return its most general unifier, if exists. Return NONE otherwise.

46.4 See Also

See also TY and Infer.

47 WellDone

47.1 Synopsis

structure WellDone

The 'welldone' representation for CPL programs. This is the most important representation and is used by the optimizer and the interpretor.

47.2 Signature

```
type priority = real (* priority for accessing a server *)
type vno = int (* variable number *)
type tno = int (* user-defined type number *)
type sno = int (* server number *)
type pno = int (* primitive number *)
type lno = int (* label number *)

datatype SYN
  =  Unit | Integer of int | Real  of real           (* base type *)
  |  String of string |  Bool of bool
  |  User of tno * SYN
  |  Record of SYN Record.record                     (* record *)
  |  Proj of lno * SYN
  |  Modify of lno * SYN * SYN
  |  Jump of (lno -> CO.CO -> CO.CO) ref * lno * SYN
           (* Used internally by the system.
            * to make projection more efficient. *)
  |  Variant of lno * SYN                            (* variant *)
  |  Case of SYN * (lno * SYN) Record.record
  |  Otherwise of SYN * (lno * SYN) Record.record * SYN
  |  IfThenElse of SYN * SYN * SYN
  |  Fun of vno * SYN | Apply of SYN * SYN           (* function *)
  |  Primitive of pno | Variable of vno              (* primitive *)
  |  Nullary of CO.CO
  |  Unary of (CO.CO -> CO.CO) * SYN
  |  Binary of (CO.CO * CO.CO -> CO.CO) * SYN * SYN
           (* Nullary, unary, and binary are used
            * internally by the system
            * to cache static objects. *)
  |  Error of SYN                                    (* exception *)
  |  Eq of SYN * SYN | Leq of SYN * SYN              (* ordering *)
  |  Lt of SYN * SYN
  |  EmptySet | SngSet of SYN                        (* collections *)
  |  UnionSet of SYN * SYN
  |  ExtSet of SYN * vno * SYN
  |  EmptyBag |  SngBag of SYN
  |  UnionBag of SYN * SYN
```

```
  |  ExtBag of SYN * vno * SYN
  |  EmptyList |  SngList of SYN
  |  UnionList of SYN * SYN
  |  ExtList of SYN * vno * SYN
  |  ExtSB of SYN * vno * SYN |  ExtSL of SYN * vno * SYN
  |  ExtLS of SYN * vno * SYN |  ExtLB of SYN * vno * SYN
  |  ExtBL of SYN * vno * SYN |  ExtBS of SYN * vno * SYN
  |  llExtSet of SYN * vno * SYN                     (* parallelism *)
  |  llUnionSet of SYN * SYN
  |  llExtBag of SYN * vno * SYN
  |  llUnionBag of SYN * SYN
  |  llExtList of SYN * vno * SYN
  |  llUnionList of SYN * SYN
  |  llExtSB of SYN * vno * SYN | llExtBS of SYN * vno * SYN
  |  llExtLB of SYN * vno * SYN | llExtBL of SYN * vno * SYN
  |  llExtLS of SYN * vno * SYN | llExtSL of SYN * vno * SYN
  |  lExtSet of SYN * vno * SYN                      (* laziness *)
  |  lUnionSet of SYN * SYN
  |  lExtBag of SYN * vno * SYN
  |  lUnionBag of SYN * SYN
  |  lExtList of SYN * vno * SYN
  |  lUnionList of SYN * SYN
  |  lExtSB of SYN * vno * SYN | lExtBS of SYN * vno * SYN
  |  lExtBL of SYN * vno * SYN | lExtLB of SYN * vno * SYN
  |  lExtLS of SYN * vno * SYN | lExtSL of SYN * vno * SYN
  |  lllExtSet of SYN * vno * SYN                    (* parallel-laziness *)
  |  lllUnionSet of SYN * SYN
  |  lllExtBag of SYN * vno * SYN
  |  lllUnionBag of SYN * SYN
  |  lllExtList of SYN * vno * SYN
  |  lllUnionList of SYN * SYN
  |  lllExtSB of SYN * vno * SYN | lllExtBS of SYN * vno * SYN
  |  lllExtBL of SYN * vno * SYN | lllExtLB of SYN * vno * SYN
  |  lllExtLS of SYN * vno * SYN | lllExtSL of SYN * vno * SYN
  |  Read of sno * * priority * SYN                  (* communication *)

val FreshVar : unit -> vno
val ResetVar : unit -> unit
```

47.3 Description

- SYN

  The welldone representation of CPL programs.

- FreshVar ()

  Return a fresh variable number, by incrementing a counter.

- ResetVar ()

  Reset the variable counter to zero.

47.4 See Also

See also Rare and MediumRare for the other two representations of CPL programs.

Part III

Output and Exchange Formats

48 Introduction

49 CPL Standard Exchange Format

The CPL Standard Exchange Format is a self-describing format for laying out complex objects in a data stream. It has a basic form and a compressed form. The important property of the format is that each data type is given an unambiguous encoding. As a result, the data items canbe parsed without reference to any schema.

49.1 The Basic Form

The basic form of the exchange format is given in the table below. Punctuation/indentation marks such as colon, commas, tabs, newlines, and spaces are not significant. The semicolon is used to indicate the end of a complex object.

CPL Type	Data Layout	Remarks
Unit	`()`	
Booleans	`true` `false`	

Numbers	`123`
	`123.123`
	`~123`
	`~123.123`
		Positive numbers
		Negative numbers

Strings	`"a string"`	String is put inside double quotes
Records	`(#`l_1`:` O_1, $\vdots$, `#`l_n`:` O_n`)`	Record is put inside round brackets. The label-value triplets enumerate the fields of the record
Variants	`<#`l`:` O`>`	Variant is put inside angle brackets
Sets	`{` O_1, $\vdots$, O_n`}`	Set is put inside curly brackets
Bags	`{|` O_1, $\vdots$, O_n `|}`	Bag is put inside curly-bar brackets
Lists	`[` O_1, $\vdots$, O_n`]`	List is put inside square brackets
User-defined types	`longitude "50E"`	User-defined type is preceded by its name
Errors	`error "it goofed"`	Error message is preceded by `error`

Here is an example layout of a complex object:

```
{(#history: {},
```

```
  #name: "jim"),
 (#history: {(#salary: 2100,
              #date: "july 1993"),
             (#salary: 2000,
              #date: "june 1993")},
  #name: "tom")}
```

Here is the same object laid out in an equally acceptable way:

```
{(#history: {}, #name: "jim"),
 (#history: {(#salary: 2100, #date: "july 1993"),
 (#salary: 2000, #date: "june 1993")}, #name: "tom")}
```

49.2 The compressed Form

The compressed form of the exchange format adds a few shorthands to the basic form. Basically it allows the definition and use of shorthands.

There are three categories of shorthands: labels, records, and user-defined types. In each category, upto a maximum of 255 shorthands can be defined simultaneously within a data stream. However, any shorthand can be redefined at any time within a data stream.

The shorthand for a long label is defined by replacing an occurrence of the label `#`l in the data stream by

$$\texttt{|DEFL}c\ \texttt{\#}l$$

Note the space between c and the label. Here c is any single character except the newline character. From that point onward, all occurrences of

$$\texttt{|L}c$$

are interpreted as `#`l.

For example, the earlier complex object is equivalent to the one below.

```
{(|DEFL1 #history: {}, |DEFL4 #name: "jim"),
 (|L1: {(|DEFL2 #salary: 2100, |DEFL3 #date: "july 1993"),
 (|L2: 2000, |L3: "june 1993")}, |L4: "tom")}
```

The shorthand for a record is defined by replacing an occurrence of a record `(#`l_1`:` O_1, ..., `#`l_n`:` O_n`)` in the data stream by `|DEFR`, followed by c, followed by a space, followed by the field labels of the record separated by single spaces, followed by `(`, followed by the field values listed in the same order as the field labels, followed by `)`. That is,

$$\texttt{|DEFR}c\ \texttt{\#}l_1\ \ldots\ \texttt{\#}l_n\ \texttt{(}\ O_1,\ \ldots,\ O_n\ \texttt{)}$$

Here c is any single character except the newline character. From this point onward, all occurrences of

$$\texttt{|R}c\ \texttt{(}\ O'_1\ \ldots\ O'_n\ \texttt{)}$$

are interpreted as the record `(#`l_1`:` O'_1, ..., `#`l_n`:` O'_n`)`.

For example, the complex object above is equivalent to the following:

```
{ |DEFR1 #history #name ({}, "jim"),
  |R1 { |DEFR2 #salary #date (2100, "july 1993"),
        |R2 (2000, "june 1993")}, "tom")}
```

This compressed form is especially useful for transmitting relational tables. For instance, the table

```
{ (#name: "abel", #age: 20, #city: "Alaska"),
  (#name: "carol", #age: 10, #city: "California"),
  (#name: "david", #age: 40, #city: "Dakota"),
  (#name: "francis", #age: 22, #city: "Florida")}
```

can be layout much more compactly as

```
{ |DEFR1 #name #age #city ("abel", 20, "Alaska"),
  |R1 ("carol", 10, "California"),
  |R1 ("david", 40, "Dakota"),
  |R1 ("francis", 22, "Florida")}
```

Caution. In order to avoid confusing lexical analysers that use a single character lookahead between `{|`, `{|R`, and `{|DEFR`, it is advisable to insert a space after the `{`.

The shorthand for a user-defined type is defined by replacing an occurrence of the type name N by

$$\texttt{|DEFT}c\ N$$

where c is any single character except the newline character. From this point onward, all occurrences of

$$\texttt{|T}c$$

in the data stream are interpreted as N.

49.3 Advice

It is a good idea to insert a newline character into datastream every few thousand characters. This is because some parser implementation of our data exchange format reads in one entire line before doing any parsing.

49.4 See Also

See also COI, MachineO, CPLFORMAT.

50 CPLFORMAT

50.1 Synopsis

structure CPLFORMAT

This module specifies the format for printing complex object in a way that conforms to the (uncompressed) CPL Standard Exchange Format.

50.2 Signature

```
val BaseFormat : FORMAT.Format ref
val StandardCPL : FORMAT.Format
```

50.3 Description

- BaseFormat

  Set to the format for printing user-defined base types. Should be set to StandardCPL.

- StandardCPL

  The specifier for the CPL Standard Exchange Format. It is implemented as given below.

```
val StandardCPL = {
      ResultPrefix = ref "\nResult:\n\n",
      ResultPostfix = ref "\n",
      ShowType = ref true,
      ShowTime = ref true,
      ShowReadFile = ref true,
      ShowWriteFile = ref true,
      ShowMacro = ref true,
      UseRuntimeOrder = ref true,
      AlwaysFlush = ref false,
      Truncate = ref false,
      NoToTruncate = ref 0,
      IncIndent = (fn x => fn y => x ^ (StringUtil.stringMap (fn _ => " ") y)),
      SetDelimL = "{",
      SetDelimR = "}",
      SetSep = ",",
      SplitSet = ref true,
      BagDelimL = "{|",
      BagDelimR = "|}",
      BagSep = ",",
      SplitBag = ref true,
      ListDelimL = "[",
      ListDelimR = "]",
      ListSep = ",",
```

```
SplitList = ref true,
RecDelimL = "(",
RecDelimR = ")",
RecLabTrans = (fn X => (Label.Km X) ^ ": "),
SplitRecord = ref true,
FldSep = ",",
VarDelimL = "<",
VarDelimR = ">",
VarLabTrans = (fn X => (Label.Km X) ^ ": "),
BoolTrans = Bool.makestring,
StrTrans = (fn S => "\"" ^ S ^ "\""),
IntTrans = Integer.makestring,
RealTrans = Real.makestring,
UnitTrans = "()",
BaseTrans = (fn (TNo, N) => FCOO.WriteToString
   (!BaseFormat) ((COBase.GetName TNo) ^ " ") N),
FunTrans = "--- function---" }
```

50.4 See Also

See FORMAT.

51 FCOO

51.1 Synopsis

structure FCOO

Routines to output complex objects in various formats.

51.2 Signature

```
type format = FORMAT.Format
type 'a writer

val OpenWriterOnStream : format -> CIO.outstream -> unit writer
val OpenWriterOnFile   : format -> string -> unit writer
val AppendWriterOnFile : format -> string -> unit writer
val OpenWriterOnString : format -> string -> string writer
val WriteString        : 'a writer -> string -> unit
val FlushWriter        : 'a writer -> unit
val Write              : 'a writer -> CO.CO -> unit
val CloseWriter        : 'a writer -> 'a
val WriteToFile        : format -> string -> CO.CO -> unit
val WriteToStream      : format -> CIO.outstream -> CO.CO -> unit
val WriteToFile        : format -> string -> CO.CO -> string
```

51.3 Description

All routines in this module must be supplied with an output format. All writing routines required writers to be supplied. All opening routines produce writers.

- 'a writer

  A 'a writer is a tuple $(Format, Print, Flush, TidyUp, Indent)$, where $Format$ is a output format descriptor, $Print$ is a function to perform output, $Flush$ is a function to flush the output buffer, $TidyUp$ is a function to close the output buffer, and $Indent$ is a string to be used for initial indentation.

- OpenWriterOnStream F S

  Create a writer that writes to stream S using format F. This function is implemented as follow:

```
fun OpenWriterOnStream (Format : FORMAT.Format) Stream =
   (Format,                                 (* format *)
    fn S => output (Stream, S) before (
      if !(#AlwaysFlush Format)
      then flush_out Stream
      else ()),                             (* printer *)
```

```
     fn () => flush_out Stream,             (* Flush *)
     fn () => flush_out Stream,             (* tidy up *)
    "")                                     (* indentation *)
```

- OpenWriterOnFile F S

  Create a writer that writes to file S using format F.

- AppendWriterOnFile F S

  Create a writer that appends to file S using format F.

- OpenWriterOnString F S

  Create a writer that appends to the string S using format F. All strings written by this writer will be accumulated and returned when the writer is closed.

- WriteString W S

  Output the string S using writer W.

- FlushWriter W

  Flush the writer W. This makes sure all characters in the output buffer are written out.

- Write W M

  Output the complex object M using the writer W.

- CloseWriter W

  Close the writer W. If W was created by OpenWriterOnString, the result will be a string formed by concatenating all the strings written by by W in chronological order.

- WriteToFile F S M

  Output the complex object M to the file S using the format F.

- WriteToStream F S M

  Output the complex object M to the stream S using the format F.

- WriteToString F S M

  Append the complex object M to the string S using the format F.

51.4 See Also

See FORMAT, CPLFORMAT, HTMLFORMAT, and SGMLFORMAT for various formats currently implemented in Kleisli.

52 FORMAT

52.1 Synopsis

structure FORMAT

This purpose of this module is to define the type Format used for specifying output format of complex objects.

52.2 Signature

```
type Format ={
      ResultPrefix : string ref,
      ResultPostfix : string ref,
      ShowType : bool ref,
      ShowTime : bool ref,
      ShowReadFile : bool ref,
      ShowWriteFile : bool ref,
      ShowMacro : bool ref,
      UseRuntimeOrder : bool ref,
      AlwaysFlush : bool ref,
      Truncate : bool ref,
      NoToTruncate : int ref,
      IncIndent : string -> string -> string,
      SetDelimL : string,
      SetDelimR : string,
      SetSep : string,
      SplitSet : bool ref,
      BagDelimL : string,
      BagDelimR : string,
      BagSep : string,
      SplitBag : bool ref,
      ListDelimL : string,
      ListDelimR : string,
      ListSep : string,
      SplitList : bool ref,
      RecDelimL : string,
      RecDelimR : string,
      RecLabTrans : int -> string (* LabNo -> string *),
      SplitRecord : bool ref,
      FldSep : string,
      VarDelimL : string,
      VarDelimR : string,
      VarLabTrans : int -> string (* LabNo -> string *),
      BoolTrans : bool -> string,
      StrTrans : string -> string,
      IntTrans : int -> string,
      RealTrans : real -> string,
```

```
    UnitTrans : string,
    BaseTrans : int * CO.CO -> string,
    FunTrans : string }
```

52.3 Description

This record type Format specifies the parameters to be used to control the output of complex objects by FCOO routines. The purpose of the various fields of this type is given below. Some of these fields are used only by the Control module of the compiler interface.

- ResultPrefix

  Set to the string that is to be printed right before a complex object is printed. Used by Control.

- ResultPostfix

  Set to the string that is to be printed right after a complex object is printed. Used by Control.

- ShowType

  Set to *true* to print type information. Used by Control.

- ShowTime

  Set to *true* to print timing information. Used by Control.

- ShowReadFile

  Set to *true* to show the result of the CPL `readfile` command. Used by Control.

- ShowWriteFile

  Set to *true* to show the result of the CPL `writefile` command. Used by Control.

- ShowMacro

  Set to *true* to show the result of the CPL `primitive` command. Used by Control.

- UseRuntimeOrder

  Set to *true* if the complex object does not require any form of tidyin up such as duplicate elimination and sorting.

- AlwaysFlush

  Set to *true* if output should not be buffered.

- Truncate

  Set to *true* to cause sets, bags, and lists in the output to be truncated down to n elements.

- NoToTruncate

  Set this to the n to be used in Truncate above. That is, chop after this number of elements.

- IncIndent

  The function to be used for changing indentation.

- SetDelimL

  The string to be used as the open bracket for set. Note that a set is printed like this: SetDelimL *elm* SetSep ... SetSep *elm* SetDelimR.

- SetDelimR

  The string to be used as the close bracket for set.

- SetSep

  The string to be used as the separator between elements of a set.

- SplitSet

  Set to *true* to print each element of a set on a new line.

- BagDelimL

  The string to be used as the open bracket for bag.

- BagDelimR

  The string to be used as the close bracket for bag.

- BagSep

  The string to be used as the separator between elements of a bag.

- SplitBag

  Set to *true* to print each element of a bag on a new line.

- ListDelimL

  The string to be used as the open bracket for list.

- ListDelimR

  The string to be used as the close bracket for list.

- ListSep

  The string to be used as the separator between elements of a list.

- SplitList

  Set to *true* to print each element of a list on a new line.

- RecDelimL

  The string to be used as the open bracket for record. Note that a record is printed as: RecDelimL *field* FldSep ... FldSep *field* RecDelimR. A field is printed as: RecLabTrans(*label*) *value*.

- RecDelimR

  The string to be used as the close bracket for record.

- FldSep

  The string to be used as the separator between fields of a record.

- SplitRecord

  Set to *true* to print each field of a record on a new line.

- RecLabTrans

  The function to reformat the label of a record field.

- VarDelimL

  The string to be used as the open bracket for variant. A variant is printed as: VarDelimL VarLabTrans(*label*) *value* VarDelimR.

- VarDelimR

  The string to be used as the close bracket for variant.

- VarLabTrans

  The function to reformat the tag of a variant.

- IntTrans

  The function to format integers.

- RealTrans

  The function to format real numbers.

- BoolTrans

  The function to format Booleans.

- StrTrans

  The function to format strings.

- UnitTrans

  The string to be used when printing the unit object.

- FunTrans

  The string to be used when printing functions.

- BaseTrans

  The function to be used for formating user-defined base types.

52.4 See Also

See also CPLFORMAT, SGMLFORMAT, HTMLFORMAT, FCOO.

53 HTMLFORMAT

53.1 Synopsis

structure HTMLFORMAT

This module privides the specification for printing complex object in HTML format.

53.2 Signature

```
val BaseFormat : FORMAT.Format ref
val HTML : FORMAT.Format
```

53.3 Description

- BaseFormat

  The format to be used for user-defined base types. Set to HTML by default.

- HTML

  The specification for output in HTML format. Its implementation is given below.

  ```
  val HTML = {
        ResultPrefix = ref "\n",
        ResultPostfix = ref "\n<HR>\n",
        ShowType = ref false,
        ShowTime = ref false,
        ShowReadFile = ref false,
        ShowWriteFile = ref false,
        ShowMacro = ref false,
        UseRuntimeOrder = ref true,
        AlwaysFlush = ref false,
        Truncate = ref true,
        NoToTruncate = ref 50,
        IncIndent = (fn x => fn y => x ^ (StringUtil.stringMap (fn _ => " ") y)),
        SetDelimL = "<UL><LI>",
        SetDelimR = "</UL>",
        SetSep = "<LI>",
        SplitSet = ref true,
        BagDelimL = "<UL><LI>",
        BagDelimR = "</UL>",
        BagSep = "<LI>",
        SplitBag = ref true,
        ListDelimL = "<OL><LI>",
        ListDelimR = "</OL>",
        ListSep = "<LI>",
        SplitList = ref true,
  ```

```
RecDelimL = "<UL>",
RecDelimR = "</UL>",
RecLabTrans = (fn X => "<b>" ^
               (StringUtil.stringTrans ("0123456789", "          ")
                  (StringUtil.suffix(Label.Km X,1))) ^ "</b> "),
SplitRecord = ref true,
FldSep = "<br>",
VarDelimL = "<UL>",
VarDelimR = "</UL>",
VarLabTrans = (fn X => "<i>" ^ (StringUtil.suffix(Label.Km X,1))^"</i> "),
BoolTrans = Bool.makestring,
StrTrans = (fn S => S),
IntTrans = (StringUtil.stringTrans ("~", "-")) o Integer.makestring,
RealTrans = (StringUtil.stringTrans ("~", "-")) o Real.makestring,
UnitTrans = "()",
BaseTrans = (fn (TNo, N) =>
   "<DL><DT>" ^ (COBase.GetName TNo) ^
   "<DD>" ^ (FCOO.WriteToString (!BaseFormat) "" N) ^
   "</DL>"),
FunTrans = "--- function---" }
```

53.4 See Also

See FORMAT.

54 SGMLFORMAT

54.1 Synopsis

structure SGMLFORMAT

This module specifies a SGML output format for complex objects.

54.2 Signature

```
val SGML : FORMAT.Format
val BaseFormat : FORMAT.Format ref
```

54.3 Description

- BaseFormat

  Set to the format to be used for user-defined base types. Should be set to SGML here.

- SGML

  The SGML specification that conforms to the DTD below.

  ```
  <!DOCTYPE cpldata [
  <!ENTITY % univ "set|bag|list|record|variant|
                   string|num|bool|base|unit|function">
  <!ELEMENT cpldata o o (%univ;)>
  <!ELEMENT (set|list|bag) - - (elem)*>
  <!ELEMENT elem - o (%univ;)?>
  <!ELEMENT label o o (#PCDATA)>
  <!ELEMENT bool - - (#PCDATA)>
  <!ELEMENT field - o (label, (%univ;))>
  <!ELEMENT record - - (field)+>
  <!ELEMENT variant - - (field)>
  <!ELEMENT unit - o EMPTY>
  <!ELEMENT base - - (id, (%univ;))>
  <!ELEMENT id o o (#PCDATA)>
  <!ELEMENT function - o EMPTY>
  <!ELEMENT num - - (#PCDATA)>
  ]>
  ```

  It is implemented as below.

  ```
  val SGML = {
        ResultPrefix = ref "\n<cpldata>\n",
        ResultPostfix = ref "</cpldata>\n",
        ShowType = ref false,
        ShowTime = ref false,
  ```

```
ShowReadFile = ref false,
ShowWriteFile = ref false,
ShowMacro = ref false,
UseRuntimeOrder = ref true,
AlwaysFlush = ref false,
Truncate = ref false,
NoToTruncate = ref 0,
IncIndent = (fn x => fn y => x ^ (StringUtil.stringMap (fn _ => " ") y)),
SetDelimL = "<set><elem>",
SetDelimR = "</set>",
SetSep = "<elem>",
SplitSet = ref true,
BagDelimL = "<bag><elem>",
BagDelimR = "</bag>",
BagSep = "<elem>",
SplitBag = ref true,
ListDelimL = "<list><elem>",
ListDelimR = "</list>",
ListSep = "<elem>",
SplitList = ref true,
RecDelimL = "<record>",
RecDelimR = "</record>",
RecLabTrans = (fn X => "<field>" ^ (Label.Km X)),
SplitRecord = ref true,
FldSep = "",
VarDelimL = "<variant>",
VarDelimR = "</variant>",
VarLabTrans = (fn X => "<field>" ^ Label.Km X),
BoolTrans = (fn X => "<bool>" ^ (Bool.makestring X) ^ "</bool>"),
StrTrans = (fn S => "<string>" ^ S ^ "</string>"),
IntTrans = (fn X => "<num>" ^ (Integer.makestring X) ^ "</num>"),
RealTrans = (fn X => "<num>" ^ (Real.makestring X) ^ "</num>"),
UnitTrans = "<unit>",
BaseTrans = (fn (TNo, N) =>
   "<base>" ^ (COBase.GetName TNo) ^
   (FCOO.WriteToString (!BaseFormat) "" N) ^
   "</base>"),
FunTrans = "<function>" }
```

54.4 See Also

See FORMAT.

Part IV

Libraries

55 Introduction

56 ARRAY

56.1 Synopsis

structure ARRAY

This module extends Kleisli with a new CPL base type `array` that is really an abstract data type for multidimensional homogeneous arrays. These arrays are zero-based and are actually implemented by storing their elements in big one-dimensional arrays.

Notations. We write `num^k` to denote the type of a tuple of k numbers and `array(num^k,` s`)` to denote the type of k-dimensional arrays with elements of type s.

56.2 Signature

```
type tno = int (* base type number *)
type pno = int (* primitive number *)

val TNo : tno  (* The base type number of array *)

val Mk' : int list * /''1/ array -> /array(num^k, ''1)/
val Mk : int list -> /''1/ list -> /array(num^k,''1)/
val MkPNo : pno
val Km' : /array(num^k,''1)/ -> int list * /''1/ array
val Km : /array(num^k,''1)/ -> int list * /''1/ list
val KmPNo : pno

val Dim : /array(num^k,''1)/ -> int list
val DimPNo : pno
val DimKPNo : pno
val Size : /array(num^k,''1)/ -> int
val SizePNo : pno

val Jump : /array(num^k,''1)/ -> int -> /''1/
val JumpPNo : pno
val Offset : /array(num^k,''1)/ -> int list -> int
val OffsetPNo : pno
val Sub : /array(num^k,''1)/ -> int list -> /''1/
val SubPNo : pno
val CheckBound : /array(num^k,''1)/ -> int list -> bool
val CheckBoundPNo : pno

val Graph : /array(num^k,''1)/ -> /{(num^k,''1)}/
val GraphPNo : pno
val Sri : int list -> (/''1/ -> /''2/ -> /''2/) -> /''2/ -> /{(num^k,''1)}/ -> /array(num^k,''2)/
val SriPNo : pno
val Tabulate : int list -> (int list -> /''1/) -> /array(num^k,''1)/
val TabulatePNo : pno
```

```
val Subslab : /array(num^k,''1)/ -> int list -> int list -> /array(num^k,''1)/
val SubslabPNo : pno

val IndexAdd : int list -> int list -> int list
val IndexAddPNo : pno
val IndexMinus : int list -> int list -> int list
val IndexMinusPNo : pno
val IndexCheck : int list -> int list -> bool
val IndexCheckPNo : pno
val IndexOffset : int list -> int list -> int
val IndexOffsetPNo : pno

val ModifyInPlace : /array(num^k,''1)/ -> int list -> /''1/ -> unit
val MapInPlace : /array(num^k,''1)/ -> (/''1/ -> /''1/) -> unit
val ApplyInPlace : /array(num^k,''1)/ -> int list -> (/''1/ -> /''1/) -> unit
```

56.3 Description

- TNo

  Kleisli refers to the new CPL type `array` by this number.

- Mk *Dims Elems*

  Return the array whose dimensions are given by *Dims* and populate it using *Elems* in row-major order. It is used to implement the `array-mk` primitive of CPL.

- MkPNo

  Kleisli refers to the CPL primitive `array-mk` using this number.

- Mk' (*Dims*, *Elems*)

  Turn the one-dimensional ML vector *Elems* into a CPL array of dimensions *Dims*.

- Km *Array*

  Inverse of MK. Used to implement the CPL primitive `array-km`.

- KmPNo.

  Kleisli refers to the CPL primitive `array-km` using this number.

- Km' *Array*

  Inverse of Mk'.

- Dim *Array*

  Return the dimensions of *Array*. Used to implement the CPL primitive `array-dim`.

- DimPNo

  Kleisli refers to the CPL primitive `array-dim` using this number.

- DimKPNo.

  The CPL primitive `array-dimk` returns the size of the kth dimension of an array. Kleisli refers to this CPL primitive using the number DimKPNo.

- Size *Array*

  The number of elements in *Array*. Used to implement the CPL primitive `array-size`.

- SizePNo

  Kleisli refers to the CPL primitive `array-size` using this number.

- CheckBound *Array Index*

  True iff *Index* is within the bounds of *Array*. Used to implement the CPL primitive `array-checkbound`.

- CheckBoundPNo

  Kleisli refers to the CPL primitive `array-checkbound` using this number.

- Jump *Array* n

  View *Array* as a one-dimensional vector and return the nth element of the vector. Used to implement the CPL primitive `array-jump`.

- JumpPNo

  Kleisli refers to the CPL primitive `array-jump` using this number.

- Offset *Array Index*

  Given an *Index* into the k-dimensional *Array*, return equivalent index into the equivalent one-dimensional vector. Used to implement the CPL primitive `array-offset`.

- OffsetPNo

  Kleisli refers to the CPL primitive `array-offset` using this number.

- Sub *Array Index*

  Return the element of *Array* at position given by *Index*. Satisfies the axiom Sub A I $=$ Jump A (Offset A I). Used to implement the CPL primitive `array-sub`.

- SubPNo

  Kleisli refers to the CPL primitive `array-sub` using this number.

- Tabulate *Dims Generator*.

  Create an array of dimension *Dims* and populate it using the *Generator*. The *Generator* accepts an index position and returns the object that should appear in that position of the resulting array. Used to implement the CPL primitive `array-tabulate`.

- TabulatePNo

  Kleisli refers to the CPL primitive `array-tabulate` using this number.

- Subslab *Array Pivot Dims*

  Extract a slab of an array. The first input is the array. The second input is the pivot point. The last input specifies the size of the slab. Used to implement the CPL primitive `array-subslab`

- SubslabPNo

  Kleisli refers to the CPL primitive `array-subslab` using this number.

- Graph *Array*

  Return the graph of the *Array*. Used to implement the CPL primitive `array-graph`.

- GraphPNo

  Kleisli refers to the CPL primitive `array-graph` using this number.

- Sri *Dims Xform Init Graph*

  Create an array from *Graph*. It works as follow. It first creates an array A with dimensions specified by *Dims*. It initializes all elements of A to *Init*. For for each (d, x) in *Graph*, it fetches the current dth element y of A, and replaces it by $(Xform\ x\ y)$. Note that the $Xform$ above should be idempotent and commutative. Used to implement the CPL primitive `array-sri`.

- SriPNo

  Kleisli refers to the CPL primitive `array-sri` using this number.

- IndexAdd I J

  Componentwise addition of array indices. Used to implement the CPL primitive `array-index+`.

- IndexAddPNo

  Kleisli refers to the CPL primitive `array-index+` using this number.

- IndexMinus I J

  Componentwise subtraction of array indices. Used to implement the CPL primitive `array-index-`.

- IndexMinusPNo

  Kleisli refers to the CPL primitive `array-index-` using this number.

- IndexCheck *Dims Index*

  Treat *Dims* as the dimensions of an array. Return true iff *Index* is within the bounds of *Dims*. Used to implement the CPL primitive `array-indexcheck`.

- IndexCheckPNo

  Kleisli refers to the CPL primitive `array-indexcheck` using this number.

- IndexOffset *Dims Index*

  Treat *Dims* as the dimensions of an array. Return the position corresponding to *Index* when this array is viewed as a one-dimensional vector. Used to implement the CPL primitive `array-indexoffset`.

- IndexOffsetPNo

  Kleisli refers to the CPL primitive `array-indexoffset` using this number.

- ModifyInPlace *Array Index Elem*

  Destructively put *Elem* into *Array* at the place given by *Index*.

- ApplyInPlace *Array Index Xform*

  Destructively apply the function *Xform* to the element of *Array* at the position given by *Index*.

- MapInPlace *Array Xform*

  Destructively apply the function *Xform* to all elements of *Array*.

56.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

57 BAG

57.1 Synopsis

structure BAG

This module registers many functions found in COBag for use inside CPL.

57.2 Signature

```
type pno = int (* primitive number *)

val bagIndex : (/''1/ -> /''2/) * /{|''1|}/ * ((/''2/ -> /{|''1|}/) *
       (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) *
       (/''2/ -> /{|''1|}/) -> 'a) -> 'a
val bagIndex' : (/''1/ -> /''2/) * /{|''1|}/ -> ((/''2/ -> /{|''1|}/) *
       (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) * (/''2/ -> /{|''1|}/) *
       (/''2/ -> /{|''1|}/))
val bagForallPNo : pno
val bagThereisPNo : pno
val bagNumItemsPNo : pno
val bagIsEmptyPNo : pno
val bagHeadPNo : pno
val bagSru : ('a * 'a -> 'a) * (/''1/ -> 'a) * 'a -> /{|''1|}/ -> 'a
val bagSri : (/''1/ * 'a -> 'a) * 'a -> /{|''1|}/ -> 'a
val bagTrimPNo : pno
val bagSruPNo : pno
val bagSriPNo : pno
val bagHom : ('a * 'a -> 'a) * (/''1/ -> 'a) * 'a -> /{|''1|}/ -> 'a
val bagHomPNo : int
val bagFirstN : /num/ * /{|''1|}/ -> /{|''1|}/
val bagSubbagPNo : pno
val bagMonusPNo : pno
val bagInsertNPNo : pno
val bagTailPNo : pno
val bagMinPNo : pno
val bagMaxPNo : pno
val bagFirstNPNo : pno
val bagMemberPNo : pno
val bagUniquePNo : pno
val bagIndexPNo : pno
val bagIndexPNo' : pno
```

57.3 Description

- bagTrimPNo

Kleisli refers to the `bag-trim` primitive of CPL using this primitive number. This CPL primitive is annotated to be monadic idempotent.

- bagInsertNPNo

  Kleisli refers to the `bag-insertn` primitive of CPL using this primitive number.

- bagSri (I, E) B

  Structural recursion. Equivalent to COBag.Sri (SupportB.Curry I) E B. It is used to implement the `bag-sri` primitive of CPL.

- bagSriPNo

  Kleisli refers to the `bag-sri` primitive of CPL using this primitive number.

- bagSru (U, F, E) B

  Structural recursion. Equivalent to COBag.Sru (SupportB.Curry U) F E B. It is used to implement the `bag-sru` primitive of CPL.

- bagSruPNo

  Kleisli refers to the `bag-sru` primitive of CPL using this primitive number.

- bagHom (U, F, E) B

  Structural recursion. Equivalent to COBag.Hom (SupportB.Curry U) F E B. It is used to implement the `bag-hom` primitive of CPL.

- bagHomPNo

  Kleisli refers to the `bag-hom` primitive of CPL using this primitive number.

- bagIsEmptyPNo

  Kleisli refers to the `bag-isempty` primitive of CPL using this primitive number.

- bagHeadPNo

  Kleisli refers to the `bag-head` primitive of CPL using this primitive number.

- bagTailPNo

  Kleisli refers to the `bag-tail` primitive of CPL using this primitive number.

- bagFirstN (N, M)

  The first N items in bag M. Equivalent to COBag.FirstN (CONum.Km N) M. It is used to implement the `bag-firstn` primitive of CPL.

- bagFirstNPNo

  Kleisli refers to the `bag-firstn` primitive of CPL using this primitive number.

- bagThereisPNo

  Kleisli refers to the `bag-thereis` primitive of CPL using this primitive number.

- bagForallPNo

  Kleisli refers to the `bag-forall` primitive of CPL using this primitive number.

- bagNumItemsPNo

  Kleisli refers to the `bag-numitems` primitive of CPL using this primitive number.

- bagMemberPNo

  Kleisli refers to the `bag-member` primitive of CPL using this primitive number.

- bagSubbagPNo

  Kleisli refers to the `bag-subbag` primitive of CPL using this primitive number. This primitive is annotated to be eager.

- bagMinPNo

  Kleisli refers to the `bag-min` primitive of CPL using this primitive number. This primitive is annotated to be eager, idempotent, commutative, and associative.

- bagMaxPNo

  Kleisli refers to the `bag-max` primitive of CPL using this primitive number. This primitive is annotated to be eager, idempotent, commutative, and associative.

- bagMonusPNo

  Kleisli refers to the `bag-monus` primitive of CPL using this primitive number. This primitive is annotated to be eager.

- bagUniquePNo

  Kleisli refers to the `bag-unique` primitive of CPL using this primitive number.

- bagIndex' (X, Y)

  Build an index on Y using X as the indexing function. Equivalent to (COBag.Index X Y). It is used to implement the `bag-index'` primitive of CPL.

- bagIndexPNo'

  Kleisli refers to the `bag-index'` primitive of CPL using this primitive number.

- bagIndex (X, Y, F)

  Build an index on Y using X as the indexing function. Then execute F on this index. Equivalent to F (COBag.Index X Y). It is used to implement the `bag-index` primitive of CPL.

- bagIndexPNo

  Kleisli refers to the `bag-index` primitive of CPL using this primitive number.

57.4 See Also

See also Annotations, COBag, and the *Collection Programming Language Reference Manual* [24].

58 BOOL

58.1 Synopsis

structure BOOL

This module registers many functions found in COBool for use inside CPL.

58.2 Signature

```
type pno = int (* primitive no *)

val boolOrElsePNo : pno
val boolNotPNo : pno
val boolAndAlsoPNo : pno
val boolButNotPNo : pno
```

58.3 Description

- boolAndAlsoPNo

  Kleisli refers to the `andalso` primitive of CPL using this primitive number.

- boolOrElsePNo

  Kleisli refers to the `orelse` primitive of CPL using this primitive number.

- boolNotPNo

  Kleisli refers to the `not` primitive of CPL using this primitive number.

- boolButNotPNo

  Kleisli refers to the `butnot` primitive of CPL using this primitive number.

58.4 See Also

See also COBool and the *Collection Programming Language Reference Manual* [24].

59 GEN

59.1 Sypnosis

structure GEN

This module creates and exports the primitives for fixpoint, exception handling, parallelism, and sequentiality to CPL.

59.2 Signature

```
type pno = int (* primitive number *)
type vno = int (* variable number *)

val genFix  : (/'1 -> '2/ -> /'1 -> '2/) -> /'1/ -> /'2/
val genFixPNO : pno

val genHandle  : /unit -> '1/ * /string -> '1/ -> /'1/
val genHandlePNO : pno

val genBeforePNo  : pno
val genParallelPNo  : pno

val MkLet : vno -> WellDone.SYN -> WellDone.SYN -> WellDone.SYN
```

59.3 Description

- genFix f

  The function genFix is used to create the `fix` primitive in CPL. It basically returns the fixpoint of f. Its implementation is given below.

  ```
  fun genFix F N =  COFun.Km (F (COFun.Mk (genFix F))) N
  ```

- genFixPNo

  Kleisli uses this primitive number to refer to the CPL primitive `fix`.

- genHandle (F, H)

  the function genHandle is used to create the `handle` primitive in CPL. It executes f. If an error is raised, then executes H. Its implementation is given below.

  ```
  fun genHandle (F, H) =
        (COFun.Apply F COUnit.Mk handle Error.Goofed Msg =>
        COFun.Apply H (COString.Mk Msg))
  ```

- genHandlePNo

  Kleisli uses this primitive number to refer to the CPL primitive `handle`.

- genBeforePNo

  Kleisli uses this primitive number to refer to the CPL primitive `before`. Here is how that primitive is created:

```
val genBeforePNo = Primitive.Add
   "before"
   (COFun.Mk (fn XF =>
      let val (X, F) = CORecord.KmPair XF
      in COFun.Apply F X end))
   (TYI.ReadFromString "('1, '1 -> '2) -> '2")
```

- genParallelPNo

  Kleisli uses this primitive number to refer to the CPL primitive `parallel`. Here is how we created that primitive:

```
val genParallelPNo = Primitive.Add
   "parallel"
   (COFun.Mk (fn FG =>
      let val (F, G) = CORecord.KmPair FG
          val f = CML.channel ()
          val g = CML.channel ()
      in CML.spawn (fn () => CML.send (f, COFun.Apply F COUnit.Mk));
         CML.spawn (fn () => CML.send (g, COFun.Apply G COUnit.Mk));
         CORecord.MkPair (CML.accept f) (CML.accept g)
      end))
   (TYI.ReadFromString "(unit -> '1, unit -> '2) -> ('1, '2)")
val _ = Annotations.AddEager genParallelPNo
```

- MkLet X M N

  Creates a WellDone-expression equivalent to `let` $\backslash X$ `==` M `in` N. Implemented as

```
fun MkLet X E1 E2 =
   WellDone.Apply (
      WellDone.Primitive genBeforePNo,
      WellDone.Record (Record.MkPair E1 (WellDone.Fun (X, E2))))
```

59.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

60 HET

60.1 Synopsis

structure HET

This module extends Kleisli with a new CPL base type **het** that this really an abstract data type for heterogeneous objects. Functions are provided in this module to perform wildcard and pattern matching on these objects.

60.2 Signature

```
type tno = int (* base type number *)
type pno = int (* primitive number *)
type lno = int (* label number *)
type wcard = string (* wildcard expression *)

val hetTNo : tno (* the new het type *)

val het : /?/ -> /het/
val hetPNo : pno
val hetOut : /het/ -> /?/
val hetOutPNo : pno

val hetMatch : wcard -> /het/ -> /{het}/
val hetMatchPNo : pno
val hetMultiMatch : wcard -> /het/ -> /{het}/
val hetMultiMatchPNo : pno

val hetIsField : /het/ -> lno -> bool
val hetIsFieldPNo : pno
val hetIsCase /het/ -> lno -> bool
val hetIsCasePNo : pno
val hetIsSet : /het/ -> bool
val hetIsSetPNo : pno
val hetIsBag : /het/ -> bool
val hetIsBagPNo : pno
val hetIsList : /het/ -> bool
val hetIsListPNo : pno
val hetIsBool : /het/ -> bool
val hetIsBoolPNo : pno
val hetIsString : /het/ -> bool
val hetIsStringPNo : pno
val hetIsNum : /het/ -> bool
val hetIsNumPNo : pno
val hetIsUnit : /het/ -> bool
val hetIsUnitPNo : pno
val hetIsUser : /het/ -> tno -> bool
```

```
val hetIsUserPNo : pno
```

60.3 Description

- hetTNo

  This module implements an abstract data type, `het`, for heterogeneous data and adds it to CPL. Kleisli refers to the `het` type of CPL using this number.

- het N

  Return the heterogenous object equivalent to N. It is used to implement the `het-mk` primitive of CPL.

- hetPNo

  Kleisli refers to the `het-mk` primitive of CPL using this number.

- hetIsNum N

  True iff the heterogenous object N is a number. Used to implement the `het-isnum` primitive of CPL.

- hetIsNumPNo

  Kleisli refers to the `het-isnum` primitive of CPL using this number.

- hetIsBool N

  True iff N is Boolean. Used to implement the `het-isbool` primitive of CPL.

- hetIsBoolPNo

  Kleisli refers to the `het-isbool` primitive of CPL using this number.

- hetIsString N

  True iff N is a string. Used to implement the `het-isstring` primitive of CPL.

- hetIsStringPNo

  Kleisli refers to the `het-isstring` primitive of CPL using this number.

- hetIsUnit N

  True iff N is unit. Used to implement the `het-isunit` primitive of CPL.

- hetIsUnitPNo

  Kleisli refers to the `het-isunit` primitive of CPL using this number.

- hetIsSet N

  True iff N is a set. Used to implement the `het-isset` primitive of CPL.

- hetIsSetPNo

  Kleisli refers to the `het-isset` primitive of CPL using this number.

- hetIsBag N

  True iff N is a bag. Used to implement the `het-isbag` primitive of CPL.

- hetIsBagPNo

  Kleisli refers to the `het-isbag` primitive of CPL using this number.

- hetIsList N

  True iff N is a list. Used to implement the `het-islist` primitive of CPL.

- hetIsListPNo

  Kleisli refers to the `het-islist` primitive of CPL using this number.

- hetIsField N L

  True iff N is a record and has the field L. Used to implement the `het-isfield` primitive of CPL.

- hetIsFieldPNo

  Kleisli refers to the `het-isfield` primitive of CPL using this number.

- hetIsCase N L

  True iff N is a variant and has the tag L. Used to implement the `het-iscase` primitive of CPL.

- hetIsCasePNo

  Kleisli refers to the `het-iscase` primitive of CPL using this number.

- hetIsUser N T

  True iff N is a user-defined and has type T. Used to implement the `het-isuser` primitive of CPL.

- hetIsUserPNo

  Kleisli refers to the `het-isuser` primitive of CPL using this number.

- hetOut N

  The inverse of het. Used to implement the `het-km` primitive of CPL.

- hetOutPNo

  Kleisli refers to the `het-km` primitive of CPL using this number.

- hetMatch P N

  Extract the subcomponents of the heterogeneous object N according to the wildcard specifier string P. The specifier string is interpreted as follows:

Specifier P	Heterogeneous Object H	hetMatch P H
`?`	Set, bag, or list	Elements of H
`?%`	Variant with any tag	The value of H
`##`l	Record	The `#`l field of H
`%#`l	Variant with tag `#`l	The value of H
`*##l`	Anything	All `#`l fields of records anywhere in H
`*%#l`	Anything	All values of variants with tag `#`l anywhere in H

- hetMatchPNo

  Kleisli refers to the `het-match` primitive of CPL using this number.

- hetMultiMatch P M

  Extract the subcomponents of the heterogeneous object M according to the extended specifier string P. The extended specified string is a sequence of wildcard specifier strings S_1, ..., S_n separated by a dot. It works like this. It first applies S_1 to M using hetMatch. Then it applies S_2 to each of the resulting objects in the set returned by the previous step. Then it takes the union of the resulting sets. And it repeats the same processing using S_3, S_4, and so on. Here is its simple implementation:

```
fun hetMultiMatch P M = revfold
   (fn (P', M') => COSet.lExt (hetMatch P') M')
   (StringUtil.tokenize "." (P, 0))
   (COSet.Singleton M)
```

- hetMultiMatchPNo

  Kleisli refers to the `het-multimatch` primitive of CPL using this number.

60.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

61 INFO

61.1 Synopsis

structure INFO

This module implements various functions to look up system information from CPL.

61.2 Signature

```
type pno = int (* primitive number *)

val serveGetType : string -> string
val primGetType : string -> string
val primGetTypePNo : pno
val serveListAllPNo : pno
val serveGetTypePNo : pno
```

61.3 Description

- primListAllPNo

  Kleisli refers to the `info-listallprim` primitive of CPL using this number.

- primGetType S

  Return the type of the CPL primitive S. Used to implement the `info-gettypeprim` primitive of CPL.

- primGetTypePNo

  Kleisli refers to the `info-gettypeprim` primitive of CPL using this number.

- serveListAllPNo

  Kleisli refers to the `info-listallservers` primitive of CPL using this number.

- serveGetType S

  Return the argument type of the CPL server S. Used to implement the `info-getserverspec` primitive of CPL.

- serveGetTypePNo

  Kleisli refers to the `info-getserverspec` primitive of CPL using this number.

61.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

62 LIBCONFIG

62.1 Synopsis

structure LIBCONFIG

This module puts various control flags used in various library modules in one place.

62.2 Signature

```
val Prompt : string ref
```

62.3 Description

- Prompt
  Alias to SETTINGS.Prompt. Store the string to be used as the CPL interactive prompt.

62.4 See Also

See also SETTINGS.

63 LIST

63.1 Synopsis

structure LIST

This module registers many functions found in COList for use in CPL.

63.2 Signature

```
type pno = int (* primitive number *)

val listUniqueSortPNo : pno
val listSortPNo : pno
val listNumItemsPNo : pno
val SnocPNo : pno
val listSriPNo : pno
val listIndexPNo : pno
val listIndexPNo' : pno
val listIndex : (/''1/ -> /''2/) * /[''1]/ * ((/''2/ -> /[''1]/) *
      (/''2/ -> /[''1]/) * (/''2/ -> /[''1]/) * (/''2/ -> /[''1]/) *
      (/''2/ -> /[''1]/) -> 'a) -> 'a
val listIndex' : (/''1/ -> /''2/) * /[''1]/ -> ((/''2/ -> /[''1]/) *
      (/''2/ -> /[''1]/) * (/''2/ -> /[''1]/) * (/''2/ -> /[''1]/) *
      (/''2/ -> /[''1]/))
val listGenSortPNo : pno
val listFirstN : /num/ * /[''1]/ -> /[''1]/
val listSuffixPNo : pno
val listPrefixPNo : pno
val listThereisPNo : pno
val listSri : (/''1/ * 'a -> 'a) * 'a -> /[''1]/ -> 'a
val ConsPNo : pno
val listTrimPNo : pno
val listSru : ('a * 'a -> 'a) * (/''1/ -> 'a) * 'a -> /[''1]/ -> 'a
val listPostfixPNo : pno
val listHeadPNo : pno
val listUniqueSortDescPNo : pno
val listIrs : (/''1/ * 'a -> 'a) * 'a -> /[''1]/ -> 'a
val listMemberPNo : pno
val listSubListPNo : pno
val listFirstNPNo : pno
val listSruPNo : pno
val listRevPNo : pno
val listTailPNo : pno
val listIrsPNo : pno
val listIsEmptyPNo : pno
val listSortDescPNo : pno
val listForallPNo : pno
```

63.3 Description

- listTrimPNo

  Kleisli refers to the `list-trim` primitive of CPL using this primitive number. This one is annotated to be monadic idempotent.

- ConsPNo

  Kleisli refers to the `cons` primitive of CPL using this primitive number.

- SnocPNo

  Kleisli refers to the `snoc` primitive of CPL using this primitive number.

- listSri (I, E) B

  Structural recursion. Equivalent to COList.Sri (Support.Curry I) E B. It is used to implement the `list-sri` primitive of CPL.

- listSriPNo

  Kleisli refers to the `list-sri` primitive of CPL using this primitive number.

- listIrs (I, E) B

  Structural recursion. Equivalent to COList.Irs (Support.Curry I) E B. It is used to implement the `list-irs` primitive of CPL.

- listIrsPNo

  Kleisli refers to the `list-irs` primitive of CPL using this primitive number.

- listSru (U, F, E) B

  Structural recursion. Equivalent to COList.Sru (SupportB.Curry U) F E B. It is used to implement the `list-sru` primitive of CPL.

- listSruPNo

  Kleisli refers to the `list-sru` primitive of CPL using this primitive number.

- listIsEmptyPNo

  Kleisli refers to the `list-isempty` primitive of CPL using this primitive number.

- listHeadPNo

  Kleisli refers to the `list-head` primitive of CPL using this primitive number.

- listTailPNo

  Kleisli refers to the `list-tail` primitive of CPL using this primitive number.

- listFirstN (N, M)

  First N items on M. Equivalent to COList.FirstN (CONum.Km N) M. Used to implement the `list-firstn` primitive of CPL.

- listFirstNPNo

  Kleisli refers to the `list-firstn` primitive of CPL using this primitive number.

- listThereisPNo

  Kleisli refers to the `list-thereis` primitive of CPL using this primitive number.

- listForallPNo

  Kleisli refers to the `list-forall` primitive of CPL using this primitive number.

- listNumItemsPNo

  Kleisli refers to the `list-numitems` primitive of CPL using this primitive number.

- listMemberPNo

  Kleisli refers to the `list-member` primitive of CPL using this primitive number.

- listUniqueSortDescPNo

  Kleisli refers to the `list-uniquesortdesc` primitive of CPL using this primitive number. Annotated to be monadic idempotent.

- listSortDescPNo

  Kleisli refers to the `list-sortdesc` primitive of CPL using this primitive number. Annotated to be monadic idempotent.

- listUniqueSortPNo

  Kleisli refers to the `list-uniquesort` primitive of CPL using this primitive number. Annotated to be monadic idempotent.

- listSortPNo

  Kleisli refers to the `list-sort` primitive of CPL using this primitive number. Annotated to be monadic idempotent.

- listRevPNo

  Kleisli refers to the `list-rev` primitive of CPL using this primitive number.

- listGenSortPNo

  Kleisli refers to the `list-gensort` primitive of CPL using this primitive number.

- listPrefixPNo

  Kleisli refers to the `list-prefix` primitive of CPL using this primitive number.

- listSuffixPNo

  Kleisli refers to the `list-suffix` primitive of CPL using this primitive number.

- listPostfixPNo

  Kleisli refers to the `list-postfix` primitive of CPL using this primitive number.

- listSubListPNo

  Kleisli refers to the `list-sublist` primitive of CPL using this primitive number.

- listIndex' (X, Y)

  Build an index on Y using X as the indexing function. Equivalent to (COList.Index X Y). Used to implement the `list-index'` primitive of CPL.

- listIndexPNo'

  Kleisli refers to the `list-index'` primitive of CPL using this primitive number.

- listIndex (X, Y, F)

  Build an index on Y using X as the indexing function. Then execute F on this index. Equivalent to F (COList.Index X Y). Used to implement the `list-index` primitive of CPL.

- listIndexPNo

  Kleisli refers to the `list-index` primitive of CPL using this primitive number.

63.4 See Also

See also Annotations, COList, and the *Collection Programming Language Reference Manual* [24].

64 NUM

64.1 Synopsis

structure NUM

This module registers many functions found in CONum for use in CPL.

64.2 Signature

```
type pno = int (* primitive number *)

val numMinusPNo : pno
val numStringifyPNo : pno
val numMultPNo : pno
val numDivRPNo : pno
val numStringify : /num/ -> string
val numNegatePNo : pno
val numModPNo : pno
val numDivPNo : pno
val numAddPNo : pno
val numRealFromStringPNo : pno
val numIntFromStringPNo : pno
```

64.3 Description

- numRealFromStringPNo

  Kleisli refers to the primitive `num-realfromstring` of CPL using this primitive number. It converts a string into a real number.

- numIntFromStringPNo

  Kleisli refers to the primitive `num-intfromstring` of CPL using this primitive number. It converts a string into a number.

- numAddPNo

  Kleisli refers to the primitive `+` of CPL using this primitive number. Annotated to be commutative, associative, and available in SQL.

- numMinusPNo

  Kleisli refers to the primitive `-` of CPL using this primitive number. Annotated to be available in SQL.

- numMultPNo

  Kleisli refers to the primitive `*` of CPL using this primitive number. Annotated to be commutative, associative, and available in SQL.

- numModPNo

  Kleisli refers to the primitive `mod` of CPL using this primitive number.

- numDivPNo

  Kleisli refers to the primitive `div` of CPL using this primitive number. Annotated to be available in SQL.

- numDivRPNo

  Kleisli refers to the primitive `/` of CPL using this primitive number. Annotated to be available in SQL.

- numNegatePNo

  Kleisli refers to the primitive `~` of CPL using this primitive number.

- numStringify N

  Convert the number N into a string. Used to implement the `num-stringify` primitive of CPL.

- numStringifyPNO

  Kleisli refers to the primitive `num-stringify` of CPL using this primitive number.

64.4 See Also

See also Annotations, CONum, and the *Collection Programming Language Reference Manual* [24].

65 SCRIPTS

65.1 Synopsis

structure SCRIPTS

This module implement the functions to running scripts from CPL.

65.2 Signature

```
type pno = int (* primitive number *)

val scriptsUseSML : string -> unit
val scriptsUseSMLPNo : pno
val scriptsUse : string -> unit
val scriptsUsePNo : pno
val scriptsUse' : string -> unit
val scriptsUsePNo' : pno

val EvalLoop
  : (string list ref
     * (CPLI.CPLParser.svalue,CPLI.CPLParser.pos) LrParser.Token.token
         LrParser.Stream.stream ref
     * (string * CPLI.CPLParser.pos * CPLI.CPLParser.pos -> unit)
     * (unit -> unit)
     -> FORMAT.Format * (string -> unit) * (unit -> unit) * (unit -> unit)
        * string
        -> unit) ref
```

65.3 Description

- EvalLoop

  Point to the function to be used by scriptsUse to evaluate CPL scripts. By default, this is Control.EvalLoop.

- scriptsUse S

  Read and evaluate the CPL script in file S. Used to implement the `usescript` primitive of CPL.

- scriptsUsePNo.

  Kleisli refers to the `usescript` primitive of CPL using this number.

- scriptsUse' S

  Same as scriptsUse, but it loads S only if S is not already loaded. Used to implement the `usescript'` primitive of CPL.

- scriptsUsePNo'.

  Kleisli refers to the `usescript'` primitive of CPL using this number.

- scriptsUseSML *S*

  Read and evaluate the SML script in file *S*. Used to implement the `usesmlscript` primitive of CPL.

- scriptsUseSMLPNo

  Kleisli refers to the `usesmlscript` primitive of CPL using this number.

65.4 See Also

See also Control and the *Collection Programming Language Reference Manual* [24].

66 SET

66.1 Synopsis

structure SET

This module registers many functions found in the COSet module for use inside CPL.

66.2 Signature

```
type pno = int (* primitive number *)

val setTailPNo : pno
val setFirstN : /num/ * /{''1}/ -> /{''1}/
val setFirstNPNo : pno
val setSubsetN : /{''1}/ * /num/ -> /{{''1}}/
val setSubsetPNo : pno
val setMemberPNo : pno
val setIndexPNo : pno
val setIndexPNo' : pno
val setIndex : (/''1/ -> /''2/) * /{''1}/ * ((/''2/ -> /{''1}/) *
       (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/) *
       (/''2/ -> /{''1}/) -> 'a) -> 'a
val setIndex' : (/''1/ -> /''2/) * /{''1}/ -> ((/''2/ -> /{''1}/) *
       (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/) * (/''2/ -> /{''1}/) *
       (/''2/ -> /{''1}/))
val setForallPNo : pno
val setThereisPNo  : pno
val setNumItemsPNo : pno
val setIsEmptyPNo : pno
val setSubsetNPNo : pno
val setPowersetN : /{''1}/ * /num/ -> /{{''1}}/
val setPowersetNPNo : pno
val setLimitedPowerset : /{''1}/ * /num/ -> /{{''1}}/ * /{{''1}}/
val setLimitedPowersetPNo : pno
val setHeadPNo : pno
val setTrimPNo : pno
val setSru : ('a * 'a -> 'a) * (/''1/ -> 'a) * 'a -> /{''1}/ -> 'a
val setSruPNo : pno
val setSri : (/''1/ * 'a -> 'a) * 'a -> /{''1}/ -> 'a
val setSriPNo : pno
val setRoughCountPNo : pno
val setHom : ('a * 'a -> 'a) * (/''1/ -> 'a) * 'a -> /{''1}/ -> 'a
val setHomPNo  : pno
val setIntersectPNo : pno
val setDiffPNo : pno
```

66.3 Description

- setTrimPNo

  Kleisli refers to the `set-trim` primitive of CPL using this primitive number. This CPL primitive is annotated as a monadic idempotent operator.

- setSubsetN (X, N)

  The set of all subsets of X having cardinality N. Equivalent to COSet.SubsetN X(CONum.Km N). This function is used to implement the `set-subsetn` primitive of CPL.

- setSubsetNPNo

  Kleisli refers to the `set-subsetn` primitive of CPL using this primitive number.

- setPowersetN (X, N)

  The set of all subsets of X having cardinality less than or equal to N. This function is used to implement the `set-powersetn` primitive of CPL. Equivalent to COSet.PowersetN X(CONum.Km N).

- setPowersetNPNo

  Kleisli refers to the `set-powersetn` primitive of CPL using this primitive number.

- setLimitedPowerset (N, M)

  The pair of (setPowersetN(N, M), setSubsetN(N, M)). This function is used to implement the `set-limitedpowerset` primitive of CPL. Equivalent to COSet.LimitedPowerset N (CONum.Km M)

- setLimitedPowersetPNo

  Kleisli refers to the `set-limitedpowerset` primitive of CPL using this primitive number.

- setSri (I, E) B

  Structural recursion. Equivalent to COSet.Sri (SupportB.Curry I) E B. This function is used to implement the `set-sri` primitive of CPL.

- setSriPNo

  Kleisli refers to the `set-sri` primitive of CPL using this primitive number.

- setSru (U, F, E) B

  Structural recursion. Equivalent to COSet.Sru (SupportB.Curry U) F E B. This function is used to implement the `set-sru` primitive of CPL.

- setSruPNo

  Kleisli refers to the `set-sru` primitive of CPL using this primitive number.

- setHom (U, F, E) B

  Structural recursion. Equivalent to COSet.Hom (SupportB.Curry U) F E B. This function is used to implement the `set-hom` primitive of CPL.

- setHomPNo

  Kleisli refers to the `set-hom` primitive of CPL using this primitive number.

- IsEmptyPNo

  Kleisli refers to the `set-isempty` primitive of CPL using this primitive number.

- setHeadPNo

  Kleisli refers to the `set-head` primitive of CPL using this primitive number.

- setTailPNo

  Kleisli refers to the `set-tail` primitive of CPL using this primitive number.

- setFirstN (N, M)

  First N items in set M. Equivalent to COSet.FirstN (CONum.Km N) M. This function is used to implement the `set-firstn` primitive of CPL.

- setFirstNPNo

  Kleisli refers to the `set-firstn` primitive of CPL using this primitive number.

- setThereisPNo

  Kleisli refers to the `set-thereis` primitive of CPL using this primitive number.

- setForallPNo

  Kleisli refers to the `set-forall` primitive of CPL using this primitive number.

- setNumItemsPNo

  Kleisli refers to the `set-numitems` primitive of CPL using this primitive number.

- setRoughCountPNo

  Kleisli refers to the `set-roughcount` primitive of CPL using this primitive number.

- setMemberPNo

  Kleisli refers to the `set-member` primitive of CPL using this primitive number.

- setSubsetPNo

  Kleisli refers to the `set-subset` primitive of CPL using this primitive number. This CPL primitive is annotated to be eager.

- setIntersectPNo

  Kleisli refers to the `set-intersect` primitive of CPL using this primitive number. This CPL primitive is annotated to be idempotent, commutative, associative, and eager.

- setDiffPNo

  Kleisli refers to the `set-diff` primitive of CPL using this primitive number. This CPL primitive is annotated to be eager.

- setIndex' (X, Y)

  Build an index on Y using X as the indexing function. Equivalent to (COSet.Index X Y). This function is used to implement the `set-index'` primitive of CPL.

- setIndexPNo'

  Kleisli refers to the `set-index'` primitive of CPL using this primitive number.

- setIndex (X, Y, F)

  Build an index on Y using X as the indexing function. Then execute F on this index. Equivalent to F (COSet.Index X Y). This function is used to implement the `set-index` primitive of CPL.

- setIndexPNo

  Kleisli refers to the `set-index` primitive of CPL using this primitive number.

66.4 See Also

See also Annotations, COSet, and the *Collection Programming Language Reference Manual* [24].

67 SETTINGS

67.1 Synopsis

structure SETTINGS

This module implements some functions to do system reconfiguration from CPL.

67.2 Signature

```
type pno = int (* primitive number *)

val Prompt : string ref
val settingsPrompt : string -> string
val settingsPromptPNo : pno

val settingsRaiseGoofedEvenWhenNotFatal : bool -> bool
val settingsRaiseGoofedEvenWhenNotFatalPNo : pno

val settingsUseRuntimeCmpOrder : bool -> bool
val settingsUseRuntimeCmpOrderPNo : pno

val settingsTrapErr : bool -> bool
val settingsTrapErrPNo : pno

val settingsPrintRuntimeInfo : bool -> bool
val settingsPrintRuntimeInfoPNo : pno
```

67.3 Description

- settingsTrapErr X

  Set Interpretor.TrapErr to X. Return the previous value stored there. Used to implement the `settings-traperr` primitive of CPL.

- settingsTrapErrPNo

  Kleisli refers to the `settings-traperr` primitive of CPL using this number.

- Prompt

  Store the string to be used as the CPL prompt. Alias to LIBCONFIG.Prompt.

- settingsPrompt

  Set Prompt to X. Return the previous value stored there. Used to implement the `settings-prompt` primitive of CPL.

- settingsPromptPNo

  Kleisli refers to the `settings-prompt` primitive of CPL using this number.

- settingsRaiseGoofedEvenWhenNotFatal X

  Set Error.RaiseGoofedEvenWhenNotFatal to X. Return the previous value stored there. Used to implement the `settings-raisegoofedevenwhennotfatal` primitive of CPL.

- settingsRaiseGoofedEvenWhenNotFatalPNo

  Kleisli refers to the `settings-raisegoofedevenwhennotfatal` primitive of CPL using this number.

- settingsPrintRuntimeInfo X

  Set Error.PrintRuntimeInfo to X. Return the previous value stored there. Used to implement the `settings-printruntimeinfo` primitive of CPL.

- settingsPrintRuntimeInfoPNo

  Kleisli refers to the `settings-printruntimeinfo` primitive of CPL using this number.

- settingsUseRuntimeCmpOrder X

  Set Ordering.UseRuntimeOrder to X. Return the previous value stored there. Used to implement the `settings-useruntimecmporder` primitive of CPL.

- settingsUseRuntimeCmpOrderPNo

  Kleisli refers to the verb+settings-useruntimecmporder+ primitive of CPL using this number.

67.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

68 SHORTHANDS

68.1 Synopsis

structure SHORTHANDS

This structure defines some frequently used shorthands.

68.2 Signature

```
val BinOp : /('1, '2) -> '3/ -> /'1/ -> /'2/ -> /'3/
```

68.3 Description

- BinOp f M N
  Return (COFun.Kmf)(CORecord.MkPair$M$ N), giving the effect of applying f to the pair (M, N).

69 STRING

69.1 Synopsis

structure STRING

This module registers many functions found in COString for use inside CPL.

69.2 Signature

```
type pno = int (* primitive no *)

val stringReverse : string -> string
val stringIsLike : string * string -> string
val stringReversePNo : pno
val stringToLowerPNo : pno
val stringToUpperPNo : pno
val stringTokenizePNo : pno
val stringSubstringPNo : pno
val stringSpanPNo : pno
val stringTransPNo : pno
val stringIsPrefixPNo : pno
val stringQuotePNo : pno
val stringNewLinePNo : pno
val stringTabPNo : pno
val stringIsWhitePNO : pno
val stringLengthPNo : pno
val stringChrPNo : pno
val stringExplodePNo : pno
val stringSuffixPNo : pno
val stringIsSubstringPNo : pno
val stringIsWhite : string -> bool
val stringOrdPNo : pno
val stringImplodePNo : pno
val stringConcatePNo : pno
val stringCompressedCPLFromCOPNo : pno
val stringHTMLFromCOPNo : pno
val stringStdCPLFromCOPNo: pno
val stringToCOPNo: pno
```

69.3 Description

- stringReverse S

  Reverses the string S. Used to implement `string-rev` of CPL.

- stringCompressedCPLFromCOPNo

Kleisli refers to the `string-compressedcplfromco` primitive of CPL using this primitive number. The CPL primitive is for converting complex object into a string in the Compressed Kleisli Exchange Format.

- stringHTMLFromCOPNo

  Kleisli refers to the `string-htmlfromco` primitive of CPL using this primitive number. The CPL primitive is for converting complex object into a string in HTML Format.

- stringStdCPLFromCOPNo

  Kleisli refers to the `string-stdcplfromco` primitive of CPL using this primitive number. The CPL primitive is for converting complex object into a string in the Standard Kleisli Exchange Format.

- stringToCOPNo

  Kleisli refers to the `string-stdcpltoco` primitive of CPL using this primitive number. This CPL primitive is the inverse of both `string-stdcplfromco` and `string-compressedcplfromco`.

- stringReversePNo

  Kleisli refers to the `string-rev` primitive of CPL using this primitive number.

- stringConcatePNo

  Kleisli refers to the ^ primitive of CPL using this primitive number. Annotated to be associative and available in SQL.

- stringToLowerPNo

  Kleisli refers to the `string-tolower` primitive of CPL using this primitive number.

- stringToUpperPNo

  Kleisli refers to the `string-toupper` primitive of CPL using this primitive number.

- stringIsWhite S

  Test if the character S is a whitespace. That is, is S a tab, a newline, or a space. Used to implement the `string-iswhite` primitive of CPL.

- stringIsWhitePNO

  Kleisli refers to the `string-iswhite` primitive of CPL using this primitive number.

- stringIsPrefixPNo

  Kleisli refers to the `string-isprefix` primitive of CPL using this primitive number.

- stringSuffixPNo

  Kleisli refers to the `string-suffix` primitive of CPL using this primitive number.

- stringLengthPNo

  Kleisli refers to the `string-length` primitive of CPL using this primitive number.

- stringSpanPNo

  Kleisli refers to the `string-span` primitive of CPL using this primitive number.

- stringSubstringPNo

  Kleisli refers to the `string-substring` primitive of CPL using this primitive number.

- stringIsSubstringPNo

  Kleisli refers to the `string-issubstring` primitive of CPL using this primitive number.

- stringIsLike S_1 S_2

  More sophisticated substring test. Allows S_2 to be of the form "$\%p_1\%...\%p_n''$. Returns true iff p_1, ..., p_n occur in S_1 in that order. If the leading % is omitted, then S_1 must begin with p_1. If the trailing % is omitted, then S_1 must end with p_n. Used to implement `string-islike` for CPL.

- stringIsLikePNo

  Kleisli refers to the `string-islike` primitive of CPL using this primitive number.

- stringTransPNo

  Kleisli refers to the `string-trans` primitive of CPL using this primitive number.

- stringTokenizePNo

  Kleisli refers to the `string-tokenize` primitive of CPL using this primitive number.

- stringChrPNo

  Kleisli refers to the `string-chr` primitive of CPL using this primitive number.

- stringOrdPNo

  Kleisli refers to the `string-ord` primitive of CPL using this primitive number.

- stringImplodePNo

  Kleisli refers to the `string-implode` primitive of CPL using this primitive number.

- stringExplodePNo

  Kleisli refers to the `string-explode` primitive of CPL using this primitive number.

- stringNewLinePNo

  Kleisli refers to the `string-newline` primitive of CPL using this primitive number.

- stringTabPNo

  Kleisli refers to the `string-tab` primitive of CPL using this primitive number.

- stringQuotePNo

  Kleisli refers to the `string-quote` primitive of CPL using this primitive number.

69.4 See Also

See also Annotations, COString, and the *Collection Programming Language Reference Manual* [24].

70 UNSAFE

70.1 Synopsis

structure UNSAFE

This module implements some unsafe operations that can be invoked from CPL.

70.2 Signature

```
type priority = real (* server access priority *)
type pno = int (* primitive number *)

val unsafeAccessServer : string -> priority -> CO.CO -> CO.CO
val unsafeAccessServerPNo : pno

val unsafeDelServer : string -> bool
val unsafeDelServerPNo : pno

val unsafeDelPrimitive : string -> bool
val unsafeDelPrimitivePNo : pno

val unsafeAccessPrimitive : string -> CO.CO
val unsafeAccessPrimitivePNo : pno

val unsafeForce : /'1/ -> /'1/
val unsafeForcePNo : pno
val unsafeMaterialize : string -> bool
val unsafeMaterializePNo : pno
```

70.3 Description

- unsafeDelPrimitive *P*

  Delete the CPL primitive *P*. Used to implement the `delete-primitive` primitive of CPL.

- unsafeDelPrimitivePNo

  Kleisli refers to the `delete-primitive` primitive of CPL using this number.

- unsafeDelServer *S*

  Delete the CPL server *S*. Used to implement the `delete-server` primitive of CPL.

- unsafeDelServerPNo

  Kleisli refers to the `delete-server` primitive of CPL using this number.

- unsafeAccessPrimitive *P*

  Return the CPL primitive *P*. Used to implement the `access-primitive` primitive of CPL.

- unsafeAccessPrimitivePNo

  Kleisli refers to the `access-primitive` primitive of CPL using this number.

- unsafeAccessServer S

  Return the CPL server S as a function. Used to implement the `access-server` primitive of CPL.

- unsafeAccessServerPNo

  Kleisli refers to the `access-server` primitive of CPL using this number.

- unsafeForce X

  To force the complex object X to be completely evaluated. It is to be used in defining unsafeMaterialize below. It is also used to implement the `force` primitive of CPL.

- unsafeForcePNo

  Kleisli refers to the `force` primitive of CPL using this number.

- unsafeMaterialize P

  Cause the CPL primitive P to be materialized once and for all. Its implementation uses unsafeForce:

```
fun unsafeMaterialize S = (case Primitive.Peek S
   of NONE => false
   | SOME PNo => let
      val Type = Primitive.GetType PNo
      val WD = Primitive.GetMacro PNo
      val CO = unsafeForce (Interpretor.Run WD)
      val _ = Primitive.Reset PNo S CO Type
      in true
      end)
```

- unsafeMaterializePNo

  Kleisli refers to the `materialize` operation of CPL using this number.

70.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

Part V

Data Drivers

71 Introduction

72 CLUSTER

72.1 Synopsis

structure CLUSTER

Chen Jing implemented a clustering program after some discussion with Pang Hwee Hwa and myself. His program, cluster, reads lines of numbers. Numbers on the same lines are considered to be related. Numbers on two lines that share some common numbers are also considered to be related. The program simply groups related numbers into the same sets and print these sets out. The interface to this program is provided by the CLUSTER module.

72.2 Signature

```
val ClusterSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val ClusterCodes : string ref
val ClusterSvNo : int
```

72.3 Description

- `ClusterCodes`

  Points to Chen Jing's `cluster` program.

- `ClusterSvNo`

  We have pre-registered a connection to Chen Jing's program. Its name in CPL is `cluster`. Kleisli refers to it using the server number `ClusterSvNo`.

- `ClusterSv` ()

  Create a server connection to Chen Jing's `cluster` program. Return a pair of functions (*Dispatch*, *ShutDown*). Call *ShutDown*() to close the connection. Call *Dispatch Req ReplyCh ReadyCh* to send a request down this connection. The request *Req* should be a set of set of numbers. Given a set of set of numbers $\{S_1, \ldots, S_n\}$. All numbers in the same set S_i are deemed related. If $S_i \cap S_j \neq \{\}$, then numbers in S_i and S_j are deemed related. Related numbers are then grouped into clusters C_1, ..., C_m, such that any two numbers from any two clusters are not related and all numbers in the same cluster are related. Representatives r_1, ..., r_m are chosen for these clusters. Here is how a representative r for a cluster C is chosen. First r must be a member of the C. Second r must appear in at least as many S_i as any other member of C. These clusters and representatives are then returned via the *ReplyCh* channel as an objec having CPL type `{(#rep: num, #cluster: { num })}`.

72.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

73 ENTREZSVR

73.1 Synopsis

structure ENTREZSVR

This module provides access to GenBank using tools from the Entrez package developed at the National Center for Biotechnology Information [16]. This is a complex data source, where an average entry has over 12 levels of nesting and over 150 subobjects, in the ASN.1 format [17].

The implementation of this module is really a wrapper around a C program, `asncpl`, developed by Kyle Hart at the University of Pennsylvania Human Genome Center for Chromosome 22.

73.2 Signature

```
type labno = int (* label number *)
type pno   = int (* primitive number *)

val SelectLab : labno
val LevelLab  : labno
val NameLab   : labno
val ArgsLab   : labno
val PathLab   : labno
val DBLab     : labno

val EntrezCodes : string ref
val EntrezSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val AddEntrezThread : string -> int -> string
val AddEntrezThreadPNo : pno
val DeleteEntrezThread : string -> string
val DeleteEntrezThreadPNo : pno
```

73.3 Description

- SelectLab

  The label number for "*#select*".

- LevelLab

  The label number for "*#level*".

- NameLab

  The label number for "*#name*".

- ArgsLab

  The label number for "*#args*".

- PathLab

  The label number for "*#path*".

- DBLab

  The label number for "*#db*".

- EntrezCodes

  The path to the C program `asncpl`.

- EntrezSv ()

  Create a server connection to Genbank. Return a pair of functions (*Dispatch*, *Shutdown*). The connection can be shutdown later by calling *Shutdown*(). Requests can be sent to the connection by calling *Dispatch Req ReplyCh ReadyCh*, where *Req* is a complex object representing the request, *ReplyCh* is the channel to which the reply (a complex object) from GenBank should be sent, and *ReadyCh* is the channel to which the ready signal from the connection is sent. Note that *Req* must be a complex object of type (*#db* : *string*, *#select* : *string*, *#args* : *string*, *#path* : *string*).

  The implementation of this function is given below.

```
fun EntrezSv () =
   let
      (* Test existence of server codes. *)
      val _ = if System.Unsafe.SysIO.access (!EntrezCodes, [])
              then ()
              else Error.ReportFatal "ENTREZSVR: Cannot find Entrez codes."
      (* Connects to the Entrez system using Kyle's interface. *)
      val (RepStr, ReqStr) = CIO.execute (!EntrezCodes, ["-x", ";", "-i-"])
      (* Function for shutting down the server. *)
      fun ShutDown _ = (CIO.close_in RepStr; CIO.close_out ReqStr)
      (* Function to process requests. *)
      fun Dispatch Req ReplyCh ReadyCh =
         let
            (* Extract info from search request *)
            val DB = "-d " ^ (COString.Km (CORecord.Proj DBLab Req))
            val Select = StringUtil.stringTrans ("'", "\"") (
                            COString.Km (CORecord.Proj SelectLab Req))
            val Select = if Select = "" then "" else (" -s '" ^ Select ^ "'")
            val Path = " -p " ^ (COString.Km (CORecord.Proj PathLab Req))
            val Args = " " ^ COString.Km (CORecord.Proj ArgsLab Req)
         in (* Send request to Entrez. *)
            CIO.output (ReqStr, DB ^ Select ^ Path ^ Args ^ " -C\n");
            (* Send reply and signal ready. *)
            COI.ReadFromStreamLazy ReplyCh ReadyCh RepStr
         end
      in (Dispatch, ShutDown)
      end
```

- AddEntrezThread N L

Function to add L Entrez threads/connections to the system and make it available to CPL under the name N. This is implemented as

```
fun AddEntrezThread Name Level = (
      Server.ResetAllConnections (Server.Add
         Name
         EntrezSv
         (TYI.ReadFromString
            "(#db: string, #select: string, #path: string, #args: string)")
         (fn _ => TYI.ReadFromString "{''1}")
         Level);
      "Entrez server thread " ^ Name ^ " added.")
```

- AddEntrezThreadPNo

  The AddEntrezThread function is itself made available to CPL as the primitive **entrez-add**, so that one can add new Entrez connections within CPL. Kleisli can refer to this CPL primitive **entrez-add** by the primitive number AddEntrezThreadPNo. Here is how we register **entrez-add**:

```
val AddEntrezThreadPNo = Primitive.Add
      "entrez-add"
      (COFun.Mk (fn UPS =>
         let val Name = COString.Km (CORecord.Proj NameLab UPS)
             val Level = CONum.Km (CORecord.Proj LevelLab UPS)
         in COString.Mk (AddEntrezThread Name Level)
         end))
     (TYI.ReadFromString "(#name: string, #level: num) -> string")
```

- DeleteEntrezThread N

  Shutdown all Entrez connections named N. De-register them from CPL. It is implemented as

```
fun DeleteEntrezThread Name = (case Server.Peek Name
   of SOME SvNo => (
      Server.Delete SvNo;
      "Entrez thread " ^ Name ^ " deleted.")
   | NONE => "Entrez thread " ^ Name ^ " not found.")
```

- DeleteEntrezThreadPNo

  The DeleteEntrezThread function is itself registered for use within CPL as the primitive **entrez-del**. Kleisli can refer to **entrez-del** using the primitive number DeleteEntrezThreadPNo. Here is how we registered **entrez-del**:

```
val DeleteEntrezThreadPNo = Primitive.Add
   "entrez-del"
   (COFun.Mk (COString.Mk o DeleteEntrezThread o COString.Km))
   (TYI.ReadFromString "string -> string")
```

73.4 The Underlying C Program Asncpl

A short manpage of the C program `asncpl` is given below. You do need some knowledge about GenBank to understand this data driver.

```
NAME
     asncpl, asncpln - ASN.1 data retrieval using the  Collection
     Programming Language (CPL)

SYNOPSIS
     asncpl -d db -s selection [ -f format-file -g giid -i  query
     input -k linked-db -n max -p proj-ext -w weight -CDEIRV ]

     asncpln - network version (same arguments)

     Execute "asncpl -" to see short description of command  line
     arguments.


DESCRIPTION
     asncpl is a program for querying data from ASN.1 data  files
     provided  by  the National Center for Biotechnology Informa-
     tion (NCBI).  This includes the Entrez: Sequences  data  set
     containing MEDLINE, nucleic acid, and amino acid databases.

     The basic input to the program is a database  name  (-d),  a
     selection  expression  (-s)  that determines a subset of the
     database (via index look-ups), and  a  projection-extraction
     expression  (-p) that prunes each element of the subset.  It
     is also possible to query on an NCBI id  directly  (-g)  and
     return  either  the  document  summary (-D) for an entry, or
     related entries based on pre-computed similarity links (-k).

     The output of the program is a set of complex  objects  com-
     posed of collections (sets, multi-sets, and lists), records,
     and primitive types (integer, string, etc).  Predefined out-
     put  formats  include CPL (default), prolog, and Perl5 (-l).
     For customized output formats, see the -f option which reads
     files in the format described in "sample.fmt".

     Small letter options require a single argument  (use  single
     quotes, when necessary, for shell interpolation) and capital
     letter options require no arguments.

OPTIONS
     -d database
          Specifies which database to use: MEDLINE (ml),  nucleic
          acid (na), or amino acid (aa). na is the default.
     -k linked-database
          Specifies  which  data  similarity  links  to   follow.
```

Default is the same as the -d value. Must be used in conjunction with an explict GenInfo id (-g value).

-g geninfo-id

A GenInfo Id for an entry.

-s selection

Specifies the subset of entries to retrieve via index look-up. Uses boolean expressions of index-value pairs and the keywords "and", "or", and "butnot". Parenthesis are respected. Human hemoglobin entries could be retrieved using [-s organism "homo sapiens" and word "hemoglobin"]. Indexes are word, mesh, keyword, author, journal, organism, accession, gene, protein, ecno. Index values should be surrounded with double quotes if they contain spaces. Index values with the suffix "..." will be used to do partial matches.

-p projection-extraction

Used to prune the result by specifying the paths of desired sub-trees. Patterns can be unrooted or rooted. Rooted patterns begin with "Seq-entry" or "Medline-entry" corresponding to the database used ("na" and "aa", or "ml", respectively). Concatentating ".label" to the pattern causes the projection of a field or selection of a variant. "..label" does the same thing mapped over collections of records or variants. Unrooted patterns begin with ".typename" and are matched sequentially to each occurence of the matched type encountered in the parse. Traversal of recursive paths can be specified with "{}*". For instance, retrieval of all Bioseqs contained within an entry can be done with [-p 'Seq-entry{.set.seq-set.}*.seq'].

-i Batch query input file. Executes newline delimited queries sequentially using a single login. Query specification is the same as the command line arguments, but single quotes are neither needed or allowed. The command line arguments for the original invocation of the program are ignored except for -p and -l. "-i-" specifies stdin as input (THIS IS THE ONLY VALID ARGUMENT PRESENTLY)

-D Return only the Document Summary record for each element.

-C Compressed mode. Removes whitespace, tabs, new lines in output.

-E Results printed as a sequence of elements rather than a set.

-I Makes the indication of a "present" value implicit when printing values. For example, instead of printing the values <#some: "RNA"> and <#none: ()>, it prints the present and absent values as "RNA" and <#none: ()>, respectively.

-V Verbose mode. Displays query status and running times through stderr. Good for debugging and viewing network response times.

EXAMPLES

(1)
```
asncpl -s 'gene "globin"'
```

Retrieve nucleic acid entries for globin genes.

(2)
```
asncpl -d ml -s 'author "Georgiev GP"' -p Medline-entry.cit.title..name
```

Retrieve the title of all medline entries for author G.P. Georgiev.

FILES

.ncbirc

Configuration file for ASN.1 local and remote data access. This file is created by the NCBI program "entrezCf" (see ftp info below). Note that asncpl uses a different .ncbirc file than asncpln does. The appropriate .ncbirc file is located in the same place as the binary.

sample.fmt

Custom output format file. Contains instructions for use with -f option.

AVAILABILITY

asncpl can be retrieved by anonymous ftp at cbil.humgen.upenn.edu in pub/asnquery.

Authorization to use the Network Entrez database can be obtained by sending mail to net-info@ncbi.nlm.nih.gov. The Entrez: Sequences data files can purchased on CDROM. Send mail to info@ncbi.nlm.nih.gov for more information. NCBI development libraries are available at ftp.ncbi.nlm.nih.gov.

SEE ALSO

NCBI documentation for entrez, Nentrez, entrezCf, and the NCBI SDK

BUGS

Possible failure in long queries (i.e. >30K human nucleic acid entries) due to problem with AsnLinkType().

AUTHOR

Kyle Hart
khart@cbil.humgen.upenn.edu

74 FORMATTEDOUT

74.1 Synopsis

structure FORMATTEDOUT

This module provides a server for printing complex object from CPL using user-definable formats.

74.2 Signature

```
type svno : int (* server number *)

val  FormSvGen : FORMAT.Format -> unit
       -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val HTMLSv :  unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val HTMLSvNo : svno

val Formats : (string * FORMAT.Format) list ref
val AddFormats : string * FORMAT.Format -> unit
val DelFormat : string -> unit
val FindFormat : string -> FORMAT.Format
val DynamicFormSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val DynamicFormSvNo : svno
```

74.3 Description

- FormSvGen *F* ()

  Create an output server that writes using the format *F*. Return a pair (*Dispatch*, *Shutdown*). The output server can be shutdown later by calling *Shutdown*(). Output requests can be sent to this output server by calling *Dispatch Req ReplyCh ReadyCh*, where *Req* is a complex object representing the request, *ReplyCh* is the channel to which the reply from the output server is sent, and *ReadyCh* is the channel to which the server ready signal is sent. The request *Req* must be a complex object of CPL type $(string, string, '1)$, where the first component is taken to be the name of the output file, the second component is taken to be the type of the object to be printed (this component is not really used by the system, so you can set it to any string), and the third component is the object to be printed.

  The implementation parallels that of ENTREZSV.EntrezSv and is given below.

  ```
  fun FormSvGen (Format: FORMAT.Format) () =
     let
        (* Function  to shutdown server. *)
        fun Shutdown _ = ()
        (* Function to process requests. *)
        fun Dispatch Req ReplyCh ReadyCh =
           let (* Extract info from write request. *)
  ```

```
            val (File, Typ, Obj) = CORecord.KmTriple Req
            val File = COString.Km File
            val Typ = COString.Km Typ
         in (* Write object to file *)
            if File = ""
            then FCOO.WriteToStream Format CIO.std_out Obj
            else FCOO.WriteToFile Format File Obj;
            (* Send reply and signal ready. *)
            CML.send (ReplyCh, COString.Mk
               ("Object of type " ^ Typ ^" written to file "^File^"."));
            CML.send (ReadyCh, ())
         end
   in (Dispatch, Shutdown)
   end
```

- HTMLSv

  Same as FormSvGen HTMLFORMAT.HTML. Makes an output server that uses the HTML format.

- HTMLSvNo

  An output server **htmlout** has been registered for CPL. Kleisli refers to **htmlout** using the server number HTMLSvNo. We also annotated it as a local server. How is how we registered this server:

```
val HTMLSvNo = Server.Add
   "htmlout"
   HTMLSv
   (TYI.ReadFromString "(string, string, ''1)") (* filename x unused x output *)
   (fn _ => TYI.ReadFromString "string")    (* reply is string *)
   1                                        (* 1 server is enough. *)
val _ = Annotations.AddLocalServer HTMLSvNo (* Annotate as local server. *)
```

- Formats

  A list of output formats recognized by DynamicFormSv.

- AddFormats (N, F).

  Add the output format F to Formats. Set its name to N.

- DelFormat N

  Remove the output format named N from Formats.

- FindFormat N

  Return the output format named N. If no format is named N, return CPLFORMAT.StandardCPL.

- DynamicFormSv ()

  Create an output server that uses dynamically specified formats. Return a pair of function $(Dispatch, Shutdown)$. To shutdown the server later call $Shutdown()$. To send a request to it later call $Dispatch\ Req\ ReplyCh\ ReadyCh$. The request Req must have type $(string, string, '1)$, where the first string is the output file name, the second string is the name of the output format, and the third component is the output object. It is implemented as

```
fun DynamicFormSv () =
   let (* Function for closing server. *)
      fun Shutdown _ = ()
      (* Function to process requests. *)
      fun Dispatch Req ReplyCh ReadyCh =
         let (* Extract info from write request. *)
            val (File, Id, Obj) = CORecord.KmTriple Req
            val File = COString.Km File
            val Id = COString.Km Id
         in (* Output *)
            if File = ""
            then FCOO.WriteToStream (FindFormat Id) CIO.std_out Obj
            else FCOO.WriteToFile (FindFormat Id) File Obj;
            (* Signal ready. *)
            CML.send (ReplyCh, COString.Mk
               ("Object written to file "^File^" in format "^Id^"."));
            CML.send (ReadyCh, ())
         end
   in (Dispatch, Shutdown)
   end
```

- DynamicFormSvNo

  A dynamic output server **dynamicform** has been registered for CPL. Kleisli refers to it using the server number DynamicFormSvNo. We also annotated it as a local server. Here is how we did the registration:

```
val DynamicFormSvNo = Server.Add
   "dynamicform"
   DynamicFormSv
   (TYI.ReadFromString "(string, string, ''1)") (* filename x type x output *)
   (fn _ => TYI.ReadFromString "string")        (* reply is string *)
   1                                            (* 1 server is enough. *)
val _ = Annotations.AddLocalServer DynamicFormSvNo (* Annotate as local *)
```

74.4 See Also

See FORMAT, HTMLFORMAT, CPLFORMAT, and SGMLFORMAT for information on output formats. See FCOO for the formatted output routines used by output servers above.

75 GAPBLAST

75.1 Synopsis

structure GAPBLAST

Altschul et. al. [3] announced their next generation of BLAST programs called Gapped BLAST and PSI-BLAST. Wang Jiren wrote the Kleisli/CPL driver to them The interface to this driver is in the module GAPBLAST. It is very similar to the interface to WU-BLAST in the module LocalBLAST.

75.2 Signature

75.3 Description

75.4 See Also

See also LocalBLAST.

76 HTTPCALL

76.1 Synopsis

structure HTTPCALL

I have developed a simple driver in Perl for talking to HTTP servers. This module implements an interface to this driver.

76.2 Signature

```
val lineSvNo : int
val stringSvNo : int
val stringSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val lineSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val HTTPCodes : string ref
```

76.3 Description

- stringSv ()

  Create a server for making HTTP calls to World Wide Web sites. Return a pair $(Dispath, Shutdown)$. Call $Shutdown()$ to shutdown the server. Call *Dispatch Req ReplyCh ReadyCh* to make HTTP calls. The input request *Req* must be a tuple $(Machine, Port, Method, Path, Args)$, where $Machine$ is the name or IP address of the World Wide Web site to call, *Port* is the port to connect to, $Method$ must be one of the two HTTP methods "GET" or "POST", *Path* is the path of the page to access, and *Args* is a string of arguments in the usual HTTP encoding to be passed to the web site.

- stringSvNo

  A server `httpcall-string` for making HTTP calls to World Wide Web sites has been registered for CPL. Kleisli refers to it using the server number stringSvNo. It is annotated as a local server.

- lineSv ()

  Same as stringSv, but this one returns result as a list of strings, one for each line in the result.

- lineSvNo

  A server `httpcall-line` for making HTTP calls to World Wide Web sites has been registered for CPL. Kleisli refers to it using the server number lineSvNo. It is annotated as a local server. This version returns result as a list of strings, one for each line in the result.

- HTTPCodes

  The path to the Perl script of the driver. It is an alias of SVRCONFIG.TelnetCodes.

77 LOCALBLAST

77.1 Synopsis

structure LOCALBLAST

Warren Gish's Washington University BLAST or WU-BLAST2 is a software suite for rapid and sensitive similarity searches of protein and nucleotide sequence databases [2, 11]. This suite includes the following programs: `setdb`—create a protein sequence database for BLASTing, `pressdb`—create a DNA sequence database for BLASTing, `blastp`—BLAST a protein sequence against a protein sequence database, `blastn`—BLAST a DNA sequence against a DNA sequence database, `blastx`—BLAST a DNA sequence translated in all reading frames against a protein sequence database, `tblastn`—BLAST a protein sequence against a DNA sequence database dynamically translated in all reading frames, `tblastx`—BLAST the six-frame translations of a DNA sequence against the six-frame translations of a DNA sequence database.

The driver for WU-BLAST2 consists of two Perl programs, `blastsummary` and `blastdetail`, which return summary of BLAST hits and full detail of BLAST hits respectively. This module interfaces to the driver for WU-BLAST2.

77.2 Signature

```
val SeqLab : lno
val DBLab : lno
val TitleLab : lno
val IdLab : lno
val AccessionLab : lno
val NameLab : lno

val PressDB : string ref
val SetDB : string ref
val TBlastN : string ref
val BlastX : string ref
val BlastP : string ref
val BlastN : string ref
val BlastSummary : string ref
val BlastDetail : string ref
val TBlastX : string ref

val SummaryType : string
val DetailType : string

val AddDoit : string -> string ref -> string -> string ref -> string -> string
val AddDoitPNo : string -> string ref -> string -> string ref -> int
val DoitGen : string ref -> string ref -> string -> unit
      -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val OutSvGen : string ref -> unit
```

```
      -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val TBLASTNDetailPNo : pno
val TBLASTXDetailPNo : pno
val BLASTXSummaryPNo : pno
val BLASTPSummaryPNo : pno
val BLASTNSummaryPNo : pno
val TBLASTNSummaryPNo : pno
val TBLASTXSummaryPNo : pno
val BLASTXDetailPNo : pno
val BLASTPDetailPNo : pno
val BLASTNDetailPNo : pno

val PressDBSvNo : svno
val SetDBSvNo : svno
```

77.3 Description

- `SeqLab`

  The label number for #*seq*.

- `DBLab`

  The label number for #*db*.

- `TitleLab`

  The label number for #*title*.

- `IdLab`

  The label number for #*id*.

- `AccessionLab`

  The label number for #*accession*.

- `NameLab`

  The label number for #*name*.

- `PressDB`

  The path to the `pressdb` program in the WU-BLAST2 suite.

- `SetDB`

  The path to the `setdb` program in the WU-BLAST2 suite.

- `TBlastN`

  The path to the `tblastn` program in the WU-BLAST2 suite.

- `BlastX`

  The path to the `blastx` program in the WU-BLAST2 suite.

- `BlastP`

  The path to the `blastp` program in the WU-BLAST2 suite.

- `BlastN`

  The path to the `blastn` program in the WU-BLAST2 suite.

- `TBlastX`

  The path to the `tblastx` program in the WU-BLAST2 suite.

- `BlastSummary`

  The path to the `blastsummary` program in the driver for WU-BLAST2.

- `BlastDetail`

  The path to the `blastsummary` program in the driver for WU-BLAST2.

- `SummaryType`

  The CPL type of output produced by `blastsummary`. It looks like this: `{(#uid: num, #accession: string, #title: string, #score: num, #pscore: num, #identities: num, #positives: num)}`.

- `DetailType`

  The CPL type of output produced by `blastdetail`. It looks like this: `{(#uid: num, #accession: string, #title: string, #hits: [(#score: num, #pscore: num, #identities: num, #positives: num, #querystart: num, #queryend: num, #subjectstart: num, #subjectend: num, #alignment: string)])}`.

- `AddDoit`

- `AddDoitPNo`

- `DoitGen`

- `OutSvGen`

- `PressDBSvNo`

  A server to make a connection to the `pressdb` program in the WU-BLAST2 suite has been registered for CPL as `localblast-pressdb`. Kleisli refers to it using the server number `PressDBSvNo`.

- `SetDBSvNo`

  A server to make a connection to the `setdb` program in the WU-BLAST2 suite has been registered for CPL as `localblast-setdb`. Kleisli refers to it using the server number `SetDBSvNo`.

- `TBLASTXSummaryPNo`

  A function to make a connection to the `tblastx` program in the WU-BLAST2 suite and return hit summaries has been registered for CPL as `localblast-tblastx`. Kleisli refers to it using the server number `TBLASTXSummaryPNo`.

- `BLASTXSummaryPNo`

  A function to make a connection to the `blastx` program in the WU-BLAST2 suite and return hit summaries has been registered for CPL as `localblast-blastx`. Kleisli refers to it using the server number `BLASTXSummaryPNo`.

- `BLASTPSummaryPNo`

  A function to make a connection to the `blastp` program in the WU-BLAST2 suite and return hit summaries has been registered for CPL as `localblast-blastp`. Kleisli refers to it using the server number `BLASTPSummaryPNo`.

- `BLASTNSummaryPNo`

  A function to make a connection to the `blastn` program in the WU-BLAST2 suite and return hit summaries has been registered for CPL as `localblast-blastn`. Kleisli refers to it using the server number `BLASTNSummaryPNo`.

- `TBLASTNSummaryPNo`

  A function to make a connection to the `tblastn` program in the WU-BLAST2 suite and return hit summaries has been registered for CPL as `localblast-tblastn`. Kleisli refers to it using the server number `TBLASTNSummaryPNo`.

- `TBLASTNDetailPNo`

  A function to make a connection to the `tblastn` program in the WU-BLAST2 suite and return full hit detail has been registered for CPL as `localblast-tblastn-detail`. Kleisli refers to it using the server number `TBLASTNDetailPNo`.

- `TBLASTXDetailPNo`

  A function to make a connection to the `tblastx` program in the WU-BLAST2 suite and return full hit detail has been registered for CPL as `localblast-tblastx-detail`. Kleisli refers to it using the server number `TBLASTXDetailPNo`.

- `BLASTXDetailPNo`

  A function to make a connection to the `blastx` program in the WU-BLAST2 suite and return full hit detail has been registered for CPL as `localblast-tblastx-detail`. Kleisli refers to it using the server number `BLASTXDetailPNo`.

- `BLASTPDetailPNo`

  A function to make a connection to the `blastp` program in the WU-BLAST2 suite and return full hit detail has been registered for CPL as `localblast-blastp-detail`. Kleisli refers to it using the server number `BLASTPDetailPNo`.

- `BLASTNDetailPNo`

  A function to make a connection to the `blastn` program in the WU-BLAST2 suite and return full hit detail has been registered for CPL as `localblast-blastn-detail`. Kleisli refers to it using the server number `BLASTNDetailPNo`.

77.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

78 SCOPSVR

78.1 Synopsis

structure SCOPSVR

Steve Brenner passed me a driver to his SCOP database [15] when he visited me in July 1997. The SCOP database is classification of protein families derived from sequence as well as structural similarity. Each entry is SCOP is identified by a sid string such as "d1sert_" and corresponds to a domain in a PDB entry. These entries are organized into a classification hierarchy. Each node in the hierarchy is identified by a node number such as "1.010.002.001.002.003". Several sid's can live in the same node, because there are classified to belong to the same family. Associated with each sid are the name of the PDB source of that entry such as "1ser"; the region that entry corresponds to in the PDB source such as "t:"; the corresponding node number of the SCOP hierarchy; the type and description of the entire path from the root to that node of the SCOP hierarchy. This module interfaces to Steve's driver.

78.2 Signature

```
val SidInfoLab : labno
val SidAllInfoLab : labno
val LsSidLab : labno
val NumSidLab : labno
val PDBSidLab : labno

val ScopDir : string ref
val ScopCodes : string ref
val AddScopThread : string -> string
val DeleteScopThread : string -> string
val DeleteScopThreadPNo : int
val ScopGen : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val AddScopThreadPNo : int
```

78.3 Description

- SidInfoLab

  The label number for "*#sidinfo*".

- SidAllInfoLab

  The label number for "*#sidallinfo*".

- LsSidLab

  The label number for "*#lssid*".

- NumSidLab

  The label number for "*#numsid*".

- PDBSidLab

  The label number for "*#pdbsid*".

- ScopDir

  The path to the SCOP database. It is an alias of SVRCONFIG.ScopDir.

- ScopCodes

  The path to Brenner's driver for SCOP. It is an alias of SVRCONFIG.ScopCodes.

- AddScopThread *Name*

  Make a connection to the SCOP driver and name the connection *Name*.

- AddScopThreadPNo

  The AddScopThread function is made available to CPL as the primitive `scop-add`. Kleisli refers to it using the primitive number AddScopThreadPNo.

- DeleteScopThread

  Remove all connections named *Name* to the SCOP driver.

- DeleteScopThreadPNo

  The DeleteScopThread function is made available to CPL as the primitive `scop-del`. Kleisli refers to it using the primitive number DeleteScopThreadPNo.

- ScopGen ()

  Make a connection to the SCOP driver. Return a pair of functions (*Dispatch*, *Shutdown*). Call *Shutdown*() to disconnect. Call *Dispatch Req ReplyCh ReadyCh* to send queries to SCOP. Here *Req* is acomplex object representing the SCOP query, *ReplyCh* is the channel to which the driver should send its reply, and *ReadyCh* is the channel to which the driver should send its ready signal. *Req* can take five possible forms, corresponding to the five different services provided by the SCOP driver:

  - COVariant.Mk SidInfoLab (COString.Mk *sid*).

    This requests the SCOP driver to return information on the SCOP protein domain identified by *sid*.
  - COVariant.Mk SidAllInfoLab COUnit.Mk.

    This requests the SCOP driver to return information on all SCOP domains.
  - COVariant.Mk NumSidLab (COString.Mk N).

    This requests the SCOP driver to return all the sid's in the class identified by the hierarchy number N.
  - COVariant.Mk PDBSidLab (COString.Mk N).

    This requests the SCOP driver to return all the sid's that corresponds to protein domains in the PDB entry named N.
  - COVariant.Mk LsSidLab COUnit.Mk.

    This requests the SCOP driver to return all sid's in the SCOP database.

78.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

79 SEGFILTER

79.1 Synopsis

structure SEGFILTER

Wootton and Federhen [28] implemented the SEG program for identifying and masking segments of low compositional complexity in amino acid sequences. The interface to this program is in this module.

79.2 Signature

```
val SegFilter : string ref
val SegSvNo : svno
val SegSv :unit ->(CO.CO ->CO.CO CML.chan ->unit CML.chan -> unit)* ('a ->unit)
```

79.3 Description

- SegFilter

  The path to the driver for the SEG program. The driver was developed by Jiren Wang.

- SegSv ()

  Create a connection to the SEG program using Jiren's driver. Return a pair of functions (*Dispatch*, *Shutdown*). Call *Shutdown*() to close the connection. Call *Dispatch Req ReplyCh ReadyCh* to make a SEG sequence complexity analysis request. *Req* is your amino acid sequence. *ReplyCh* is the channal SEG sends its reply. The reply has CPL type `[(#highcomplexity: bool`, `#start: num`, `#end: num`, `#seq: string)]`. *ReadyCh* is the channel the SEG program signals its readiness.

- SegSvNo

  We have pre-registered a SEG server connection `segfilter` for CPL. Kleisli refers to it using the server number SegSvNo.

79.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

80 SEQINDEXSVR

80.1 Synopsis

structure SEQINDEXSVR

My colleague, Hwee Hwa Pang, at the Institute of Systems Science developed SeqIndex, a suite of programs for indexing and matching long strings like amino acid and DNA sequences. The SeqIndex suite currently consists of the following five programs: loadseq—for creating a sequence index; scanseq—for finding sequences having patterns specified by a regular expression, modulo edit distance; signature—for detecting conserved patterns in sequences; duplicate—for detecting repeated substrings modulo edit distance in sequences; and dnaduplicate—for detecting tandem and inverted repeats modulo edit distance in DNA sequences. The interface to the SeqIndex suite is implemented in this module.

80.2 Signature

```
val ScanSeqCodes      : string ref
val LoadSeqCodes      : string ref
val SignatureCodes    : string ref
val PrositeScanCodes  : string ref
val PrositeDB         : string ref
val DuplicateCodes    : string ref
val DNADuplicateCodes : string ref

val AddScanSeqThreadPNo : int
val AddScanSeqThread : string -> string -> int -> string
val ScanSeqGen : string -> unit
   -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val AddPrositeScanThread : string -> string -> int -> string
val AddPrositeScanThreadPNo : int
val PrositeScanGen : string -> unit
   -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val AddDuplicateThread : string -> bool -> bool -> num -> string
val AddDuplicateThreadPNo : int
val DuplicateGen : bool -> bool -> unit
   -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val AddDNADuplicateThreadPNo : int
val AddDNADuplicateThread : string -> bool -> bool -> int -> string
val DNADuplicateGen : bool -> bool -> unit
   -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val SeqIndexInSvNo : int
val SeqIndexInSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val SeqIndexOutSvNo : int
```

```
val SeqIndexOutSvGen : bool -> unit ->
    (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val AddSignatureThread : string -> string -> bool -> int -> string
val AddSignatureThreadPNo : int
val SignatureGen : string -> bool -> unit
   -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
```

80.3 Description

- ScanSeqCodes

  The path to the SeqIndex/scanseq program. It is an alias of SVRCONFIG.ScanSeqCodes.

- LoadSeqCodes

  The path to the SeqIndex/loadseq program. It is an alias of SVRCONFIG.LoadSeqCodes.

- SignatureCodes

  The path to the SeqIndex/signature program. It is an alias of SVRCONFIG.SignatureCodes.

- PrositeScanCodes

  The path to the SeqIndex/prositescan program. It is an alias of SVRCONFIG.PrositeScanCodes.

- PrositeDB

  The path to the Prosite motif database. It is an alias of SVRCONFIG.PrositeDB.

- DuplicateCodes

  The path to the SeqIndex/duplicate program. It is an alias of SVRCONFIG.DuplicateCodes.

- DNADuplicateCodes

  The path to the SeqIndex/dnaduplicate program. It is an alias of SVRCONFIG.DNADuplicateCodes.

- SeqIndexInSv

  Create an input server for reading sequence index files. Return a pair (*Dispatch*, *Shutdown*). To shutdown this server, call *Shutdown*(). To input using this server call *Dispatch Req ReplyCh ReadyCh*. Here *Req* is the name of the sequence index to read, *ReplyCh* is the channel to which the result should be sent, *readyCh* is the channel to which a ready signal should be sent.

- SeqIndexInSvNo

  A sequence index input server `seqindex-in` has been registered for CPL. Kleisli refers to it using the server number SeqIndexInSvNo.

- SeqIndexOutSvGen *IsDNA* ()

  Create an output server for building sequence indices using Hwee Hwa's SeqIndex/loadseq program. If *IsDNA* is true, then use a hash function specialized for DNA sequences. Return a pair (*Dispatch*, *Shutdown*). Call *Shutdown*() to shutdown the server. Call *Dispatch Req ReplyCh ReadyCh* to build a seqence index. The *Req* is expected to have CPL type $(string, string, \{(\#uid : num, \#seq : string)\})$. The first string is the name of the sequence index to be created. The second string is ignored by the system. The third component is a set sequences to be indexed—the *#seq* field is a sequence, the *#uid* field is the unique identifier of the sequence.

- SeqIndexOutSvNo

  A sequence index output server `seqindex-out` has been registered for CPL. Kleisli refers to it using the server number SeqIndexOutSvNo.

- SeqIndexSetDBSvNo

  A sequence index output server `seqindex-setdb` has been registered for CPL. Kleisli refers to it using the server number SeqIndexSetDBSvNo.

- SeqIndexPressDBSvNo

  A sequence index output server `seqindex-pressdb` has been registered for CPL. This one is specialized for DNA sequences. Kleisli refers to it using the server number SeqIndexPressDBSvNo.

- ScanSeqGen *Index*

  Make a connection to Hwee Hwa's SeqIndex/scanseq program. Set it to use the sequence index *Index*. Return a pair $(Dispatch, Shutdown)$. Call $Shutdown()$ to close the connection. Call *Dispatch Req ReplyCh ReadyCh* to send a request to the connection. *ReplyCh* is the channel to which reply should be sent. *ReadyCh* is the channel to which ready signal should be sent. *Req* is the request, which has the CPL type: $< \#list : unit$, $\#get : num$, $\#doit : (\#restrict : bool$, $\#pattern : string$, $\#edit : num$, $\#overlap : num$, $\#favourshort : bool) >$.

  A *Req* that is a *#list* variant is a request to list all the sequences in *Index*. A *Req* that is a *#get* variant is a request to retrieve the sequence in *Index* whose *#uid* is the specified number. A *Req* that is a *#doit* variant is a request to retrieve all substrings in the index having the pattern specified in the *#pattern* field of the request. The pattern is any Perl pattern. Set the *#edit* field to the allowable edit distance. Set the *#restrict* field to false if you want to allow mismatches everywhere. Sometimes two overlapping substrings can match your pattern. Frequently you only want to see just one of such substrings. Set the *#overlap* field to N if you want to eliminate substrings that overlap at least N characters. Set the $\#favourshort$ field to true if you only want to break ties favouring shorter substrings.

- AddScanSeqThread *Name Index Level*

  Make *Level* connections to Hwee Hwa's SeqIndex/scanseq program. Name all of them *Name*. Set all of them to use the sequence index *Index*. All subsequent queries against *Index* can be sent these connections.

- AddScanSeqThreadPNo

  The AddScanSeqThread is made available to CPL as the primitive `seqindex-scanseq`. Kleisli refers to it using the primitive number AddScanSeqThreadPNo.

- PrositeScanGen *DB*

Make a connection to Hwee Hwa's SeqIndex/prositescan program. Set its motif database to DB. Return a pair $(Dispatch, Shutdown)$. Call $Shutdown()$ to disconnect. Call $Dispatch$ Req $ReplyCh$ $ReadyCh$ to if you want to use Hwee Hwa's program to figure out what motifs are in your sequence and the positions of the hits. $ReplyCh$ is the channel to which reply should be sent. $ReadyCh$ is the channel to which ready signal should be sent. Req is the request.

The Req must be a complex object having the CPL type $(\#restrict : bool, \#edit : num, \#fussy : bool, \#seq : string)$. Set the $\#seq$ field to your sequence. Set the $\#fussy$ field to true if you are only interested in rare motifs. Set the $\#edit$ field to the allowable edit distance. Set the $\#restrict$ field to false if mutations are allowed everywhere in motifs.

- AddPrositeScanThread $Name$ DB $Level$

  Make $Level$ connections to Hwee Hwa's SeqIndex/prositescan program. Name all of them $Name$. Set all of them to use the motif database DB. All subsequent queries against DB can be sent these connections.

- AddPrositeScanThreadPNo

  The AddPrositeScanThread is made available to CPL as the primitive `seqindex-prositescan`. Kleisli refers to it using the primitive number AddPrositeScanThreadPNo.

- DuplicateGen $Elim$ $OptForTime$

  Make a connection to Hwee Hwa's SeqIndex/duplicate program. Set it to eliminate subsumed repeats, if $Elim$ is true. Set it to trade space for time, if $OptForTime$ is true. Return a pair $(Dispatch, Shutdown)$. Call $Shutdown()$ to disconnect. Call $Dispatch$ Req $ReplyCh$ $ReadyCh$ if you want to use Hwee Hwa's program to figure out all the repeated patterns in your sequence. $ReplyCh$ is the channel where reply should be sent. $ReadyCh$ is the channel where ready signal should be sent. Req is your request.

  Req must be an object having CPL type $(\#min : num, \#max : num, \#incr : num, \#support : num, \#edit : num, \#space : num, \#seq : string)$. Set $\#min$ to the minimum length of the repeats you want. Set $\#max$ to the maximum length of the repeats you want. Set $\#support$ to the minimum number of times a repeat must appear. Set $\#seq$ to your input sequence. Set $\#edit$ to the allowable edit distance between two occurrences of a repeat. Set $\#space$ to the minimum separation btween two occurences of a repeat. Set $\#incr$ to the amount you want to increment the length of a repeat; for example, you can set $\#min$ to 4, $\#max$ to 10, and $\#incr$ to 2, to make the system look for repeats of lengths 4, 6, 8, 10.

- AddDuplicateThread $Name$ $Elim$ $OptForTime$ $Level$

  Make $Level$ connections to Hwee Hwa's SeqIndex/duplicate program. Name all of them $Name$. Set all of them to optimize for time if $OptForTime$ is true. Set all of them to eliminate subsumed repeats if $Elim$ is true.

- AddDuplicateThreadPNo

  The AddDuplicateThread is made available to CPL as the primitive `seqindex-duplicate`. Kleisli refers to it using the primitive number AddDuplicateThreadPNo.

- DNADuplicateGen *Elim OptForTime*

  Same behaviour as DNADuplicateGen. But connects to SeqIndex/dnaduplicate instead. As a result, it also find inverted repeats.

- AddDNADuplicateThread *Name Elim OptForTime Level*

  Make *Level* connections to Hwee Hwa's SeqIndex/dnaduplicate program. Name all of them *Name*. Set all of them to optimize for time if *OptForTime* is true. Set all of them to eliminate subsumed repeats if *Elim* is true.

- AddDNADuplicateThreadPNo

  The AddDNADuplicateThread is made available to CPL as the primitive `seqindex-dnaduplicate`. Kleisli refers to it using the primitive number AddDNADuplicateThreadPNo.

- SignatureGen *Index OptForTime*

  Make a connection to Hwee Hwa's SeqIndex/signature program. Set it to use the sequence index *Index*. Set it to trade space for time. Return a pair (*Dispatch*, *Shutdown*). Call *Shutdown*() to disconnect. Call *Dispatch Req ReplyCh ReadyCh* if you want to use SeqIndex/signature to figure out the conserved motifs of the sequences in *Index*. *ReplyCh* is the channel where result is to be sent. *ReadyCh* is the channel where ready signal is to be sent. *Req* is your request. *Req* must be a complex object having CPL type $<$ *#list* : *unit*, *#get* : *num*, *#doit* : (*#min* : *num*, *#max* : *num*, *#edit* : *num*, *#support* : *num*) $>$.

  A *Req* that is a *#list* variant is a request to retrieve all sequences in *Index*. A *Req* that is a *#get* variant is a request to retrieve the sequence in *Index* whose *#uid* is the specified number. A *Req* is that a *#doit* variant is a request to find conserved motifs in *Index*. Set *#min* to the minimum length of motifs you want to find. Set *#max* to the maximum length of motifs you want to find. Set *#support* to the minimum number of sequences in *Index* that a motif must appear. Set *#edit* to the allowable edit distance between occurrences of a motif.

- AddSignatureThread *Name Index OptForTime Level*

  Make *Level* connections to Hwee Hwa's SeqIndex/signature program. Name all of them *Name*. Set all of them to optimize for time if *OptForTime* is true. Set all of them to eliminate subsumed repeats if *Elim* is true. Set all of them to use the sequence index *Index*.

- AddSignatureThreadPNo

  The AddSignatureThread is made available to CPL as the primitive `seqindex-signature`. Kleisli refers to it using the primitive number AddSignatureThreadPNo.

80.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

81 SVRCONFIG

81.1 Synopsis

structure SVRCONFIG

Configuration files for data drivers.

81.2 Signature

```
val SybInterface   : string ref
val SybCodes       : string ref
val HTTPCodes      : string ref
val EntrezCodes    : string ref
val SmallFileLimit : int ref
val BigFileLimit   : int ref
val MaxLines       : int ref
val BlockSize      : int ref
val ScanSeqCodes   : string ref
val LoadSeqCodes   : string ref
val SignatureCodes : string ref
val PrositeScanCodes  : string ref
val PrositeDB         : string ref
val DuplicateCodes    : string ref
val DNADuplicateCodes : string ref
val PhyCodes : string ref
val PhyDir : string ref
val ScopDir : string ref
val ScopCodes : string ref
val SegFilter : string ref
val ClusterCodes : string ref
```

81.3 Description

- ClusterCodes

  The path to Chen Jing's cluster program. It is an alias to CLUSTER.ClusterCodes.

- SegFilter

  The path to Jiren Wang's driver for the SEG program. It is an alias to SEGFILTER.SegFilter.

- DNADuplicateCodes

  Same as SEQINDEXSVR.DNADuplicateCodes. The path to Hwee Hwa's SeqIndex/dnaduplicate program.

- DuplicateCodes

  Same as SEQINDEXSVR.DuplicateCodes. The path to Hwee Hwa's SeqIndex/duplicate program.

- PrositeDB

  Same as SEQINDEXSVR.PrositeDB. The path to the Prosite motif database.

- PrositeScanCodes

  Same as SEQINDEXSVR.PrositeScanCodes. The path to Hwee Hwa's SeqIndex/prositescan program.

- SignatureCodes

  Same as SEQINDEXSVR.SignatureCodes. The path to Hwee Hwa's SeqIndex/signature program.

- LoadSeqCodes

  Same as SEQINDEXSVR/LoadSeqCodes. The path to Hwee Hwa's SeqIndex/loadseq program.

- ScanSeqCodes

  Same as SEQINDEXSVR.ScanSeqCodes. The path to Hwee Hwa's SeqIndex/scanseq program.

- PhyDir

  Same as TAXONSVR.PhyDir. The path to the taxonomy database from NCBI.

- PhyCodes

  Same as TAXONSVR.PhyCodes. The path to Steve Brenner's driver for the taxonomy database.

- ScopCodes

  Same as SCOPSVR.ScopCodes. The path to Steve Brenner's SCOP driver.

- ScopDir

  Same as SCOPSVR.ScopDir. The path to the SCOP database.

- SybInterface

  Same as SYBASESVR.SybInterface. The path to the interface file for the Sybase driver.

- SybCodes

  Same as SYBASESVR.SybCodes. The path to the Sybase data driver.

- HTTPCodes

  Same as HTTPCALL.HTTPCodes. The path to the Perl driver to HTTP servers.

- EntrezCodes

  Same as ENTREZSVR.EntrezCodes. The path to the Entrez data driver.

- SmallFileLimit

  Same as STDIO.SmallFileLimit. Any file up to this size is considered small. Default is 20K.

- BigFileLimit

  Same as STDIO.BigFileLimit. Any file above this size is considered big. Default is 500K.

- MaxLines

  Same as STDIO.MaxLines. Maximum number of small files that can be cached. Default is 100.

- BlockSize

  Same as COI.BlockSize. Read big files using this block size. Default is 20K.

81.4 See Also

See also STDIO for more information on the effect of file sizes on the standard input server. See SYBASESVR for details of the Sybase data driver. See ENTREZSVR for details of the Entrez data driver. See SEQINDEXSVR for detail of Hwee Hwa's sequence indexing programs. See TAXONSVR for detail of Brenner's driver for the NCBI taxonomy database. See SCOPSVR for detail of Brenner's driver for his SCOP database.

82 STDIO

82.1 Synopsis

structure STDIO

This module implements the standard input and output servers for CPL.

82.2 Signature

```
type svno = int (* server number *)

val BlockSize : int ref
val BigFileLimit : int ref
val ReadBigFile : string -> CO.CO CML.chan -> unit CML.chan -> unit

val SmallFileLimit : int ref
val MaxLines : int ref
val ClearCache : unit -> unit
val ResetCache : int -> int -> unit
val ReadIntoCache : string -> CO.CO CML.chan -> unit CML.chan -> unit

val stdinSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val stdinSvNo : svno
val stdoutSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val stdoutSvNo : svno
val machineoutSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val machineoutSvNo : svno

val appendtofileSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val appendtofile : svno

val Stringify : string -> string
val Stringify' : string -> string
val stringifySv :  unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val stringifySvNo : svno

val IntoLines : CIO.instream -> CO.CO CML.chan -> unit CML.chan -> unit
val IntoLinesSv :  unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val IntoLinesSvNo : svno
```

82.3 Description

- BlockSize

  Same as COI.BlockSize. Read big files this number of bytes in one go.

- BigFileLimit

Same as SVRCONFIG.BigFileLimit. Any file above this size is considered big.

- ReadBigFile *File ReplyCh ReadyCh*

  Read the big file *File*. Send the result to *ReplyCh*. Send ready signal to *ReadyCh*. The file is read in big blocks, as specified in BlockSize. If the object read is a set, bag, or list, it is read lazily. That is the set, bag, or list is returned with its first few elements instantiated and the system will bring in the rest of the elements automatically when needed and only when needed. This is just a wrapper around COI.ReadFromBigFile.

- SmallFileLimit

  Same as SVRCONFIG.SmallFileLimit. Any file smaller than this size is considered small.

- MaxLines

  Same as SVRCONFIG.MaxLines. Maximum number of small files that can be cached.

- ClearCache ()

  Clear the small file cache.

- ResetCache *Limit Max*

  Set SmallFileLimit to *Limit*. Set MaxLines to *Max*. Clear the cache.

- ReadIntoCache *File ReplyCh ReadyCh*

  Read the small file *File*. Send result to *ReplyCh*. Send ready signal to *ReadyCh*. The result is cached. It is implemented using COI.ReadFromFile and therefore has eager behaviour.

- stdinSv ()

  Create a input server. Return a pair (*Dispatch*, *Shutdown*). To shutdown this server call *Shutdown*(). To input using this server call *Dispatch Req ReplyCh ReadyCh*, where *Req* is the input request, *ReplyCh* is the channel to which the result should be sent, and *ReadyCh* is the channel to which the ready signal should be sent. *Req* is assumed to be a CPL string representing input file name. A ".val" subfix is automatically added. (CPL uses the convention that local files are given a ".val" subfix and schemas of local files are given a ".typ" subfix.)

  This server automatically deploys ReadBigFile, ReadIntoCache, and COI.ReadFromFileLazy depending on the size of the input file. The implementation is given below.

```
fun stdinSv () =
   let (* Function for closing the server. *)
      val ShutDown = ClearCache
      (* Function to process requests. *)
      fun Dispatch Req ReplyCh ReadyCh =
         let (* Extract info from read request. *)
            val File = (COString.Km Req) ^ ".val"
            open System.Unsafe.SysIO
         in (* Error if file does not exist. *)
            if access (File, [])
            then  ()
            else Error.ReportFatal("STDIO.stdinSv: File "^File^" not found.");
```

```
            (* Send reply and signal ready. *)
            if fsize (PATH File) < !SmallFileLimit
            then ReadIntoCache File ReplyCh ReadyCh
            else if fsize (PATH File) < !BigFileLimit
            then COI.ReadFromFileLazy ReplyCh ReadyCh File
            else ReadBigFile File ReplyCh ReadyCh
         end
   in (Dispatch, ShutDown)
   end
```

- stdinSvNo

  An input server **stdin** has been registered for CPL. Kleisli refers to it using the server number stdinSvNo. It has also been annotated as a local server. Here is how we did the registration:

```
val stdinSvNo =
   let fun GetSchema (MediumRare.String F) =
         if System.Unsafe.SysIO.access (F ^ ".typ", [])
         then TYI.ReadFromFile (F ^ ".typ")
         else Error.ReportFatal ("STDIO.stdinSv: File "^F^".typ not found.")
       | GetSchema _ = TYI.ReadFromString "'1"
   in Server.Add
         "stdin"
         stdinSv
         (TYI.ReadFromString "string")       (* name of input file. *)
         GetSchema                           (* get reply type from schema *)
         1                                   (* run 1 servers. *)
   end
val _ = Annotations.AddLocalServer stdinSvNo
```

- stdoutSv ()

  Create a output server using COO.Format. Return a pair $(\mathit{Dispatch}, \mathit{Shutdown})$. Call $\mathit{Shutdown}()$ to shutdown the server. Call *Dispatch Req ReplyCh ReadyCh* to do output. The output request *Req* is expected to have CPL type $(\mathit{string}, \mathit{string}, ''1)$, where the first string is the name of the output file, the second string is the type of the output object (this parameter is not really used, so any type will do here), and the third component is the object to be printed. Here is the implementation:

```
fun stdoutSv () =
   let (* Function for closing the server. *)
      fun ShutDown _ = ()
      (* Function to process requests. *)
      fun Dispatch Req ReplyCh ReadyCh =
         let (* Extract info from write request. *)
            val (File, Typ, Obj) = CORecord.KmTriple Req
            val File = COString.Km File
            val Typ  = COString.Km Typ
         in (* Write Obj to .val file. *)
            COO.WriteToFile (File ^ ".val") Obj;
            (* Write Typ to .typ file. *)
            TYO.WriteToFile (File ^ ".typ") (TYI.ReadFromString Typ);
```

```
            (* Send reply and signal ready. *)
            CML.send (ReplyCh, COString.Mk
               ("Object of type "^Typ^" written to file "^File^"."));
            CML.send (ReadyCh, ())
         end
   in (Dispatch, ShutDown)
   end
```

- stdoutSvNo

  An output server **stdout** has been registered for CPL. Kleisli refers to it using the server number stdoutSvNo. We have also annotated it as a local server. Here is how we did the registration:

```
val stdoutSvNo = Server.Add
   "stdout"
   stdoutSv
   (TYI.ReadFromString "(string, string, ''1)") (* filename x type x output *)
   (fn _ => TYI.ReadFromString "string")        (* reply is string *)
   1                                            (* 1 server is enough. *)
val _ = Annotations.AddLocalServer stdoutSvNo
```

- machineoutSv ()

  Similar to stdoutSv, but uses the standard compressed CPL Exchange format.

- machineoutSvNo

  We have registered a compressed output server **machineout** for CPL. Kleisli refers to it as machineoutSvNo. It is also annotated as a local server.

- appendtofileSv ()

  Create a server for appending strings to file. Good for keeping runtime logs.

- appendtofileSvNo

  We have registered an append server **appendtofile** for CPL. Kleisli refers to it as appendtofileSvNo. It is also annotated as a local server.

- Stringify F

  Return the contents of the file F as a long string. Use Concurrent ML input-output routines.

- Stringify' F

  Return the contents of the file F as a long string. Use Standard ML input-output routines.

- stringifySv ()

  Create a server to read the contents of files as a long string. Return ($Dispatch, Shutdown$). Call $shutdown()$ to close down the server. Call $Dispatch\ Req\ ReplyCh\ ReadyCh$ to read file as a long string. Req is assumed to be the file name.

- stringifySvNo

  A "stringifcation" server **file-stringify** has been registered for CPL. Kleisli refers to it using the server number stringifySvNo. This server is annotated to be local.

- IntoLines *S ReplyCh readyCh*

  Access the stream *S* as a list of strings. Basically each line in *S* becomes an element in the list. Send the resulting list to *ReplyCh*. Send ready signal at end of stream to *ReadyCh*. The list is created lazily using a read-ahead semantics.

- IntoLinesSv ()

  Create a server for reading the contents of files into a list of strings. Return a pair $(Dispatch, Shutdown)$. Call $Shutdown()$ to shutdown the server. Call *Dispatch Req ReplyCh ReadyCh* to do input. The input request *Req* is assumed to be the input file name.

- IntoLinesSvNo

  A server file-intolines for turning a file into a list of strings has been registered for CPL. Kleisli refers to it using the server number IntoLinesSvNo. It is of course annotated as a local server.

82.4 See Also

See COI for the input routines (with eager, lazy, and big file behaviours) used by the input servers above and COO for the output routine used by the output servers oabove.

83 SYBASESVR

83.1 Synopsis

structure SYBASESVR

This module provides access to Sybase relational databases. This implementation of this module is really a wrapper around a C program, `sybcpl`, developed by Jonathan Crabtree at the University of Pennsylvania Human Genome Center for Chromosome 22.

83.2 Signature

```
type labno = int (* label number *)
type pno   = int (* primitive number *)

val SybInterface : string ref
val SybCodes : string ref

val LName   : labno
val LUser   : labno
val LPasswd : labno
val LSource : labno
val LQuery  : labno
val LLevel  : labno

val SybaseGen : string -> string -> string -> unit
       -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * (unit -> unit)
val AddSybaseThread : string -> string -> string -> string -> int -> string
val AddSybaseThreadPNo : pno
val DeleteSybaseThread : string -> string
val DeleteSybaseThreadPNo  : pno

val SybaseConnect: string -> string -> string -> string -> int -> (CO.CO -> CO.CO) * (unit -> CO.CO)
val SybaseConnectPNo : pno
val SybaseXact: string -> string -> string -> string -> int -> ((CO.CO -> CO.CO) -> 'a) -> 'a
val SybaseXactPNo : pno
```

83.3 Description

- SybInterface

  The path to a Sybase interface file. It is an alias of SVRCONFIG.SybInterface.

- SybCodes

  The path to the C program `sybcpl`. It is an alias of SVRCONFIG.SybCodes.

- LName

The label number for "*#name*".

- LUser

  The label number for "*#user*".

- LPasswd

  The label number for "*#password*".

- LSource

  The label number for "*#server*".

- LQuery

  The label number for "*#query*".

- LLevel

  The label number for "*#level*".

- SybaseGen *User Passwd Source* ()

  Connect to the server *Source*. Log in as *User* using *Passwd*. Return a pair of functions (*Dispatch*, *Shutdown*). Call *Shutdown*() to disconnect. Call *Dispatch Req ReplyCh ReadyCh* to do SQL queries. Here *Req* is a string representing a SQL query, *ReplyCh* is the channel to which the reply from the server should be sent, and *ReadyCh* is the channel where the server ready signal should be sent. This actual implementation, by Jonathan Crabtree, is given below.

```
fun SybaseGen User Passwd Source () =
   let
      (* Check if Sybase codes exists. *)
      val _ = if System.Unsafe.SysIO.access (!SybCodes, [])
              then ()
              else Error.ReportFatal "SYBASESVR: Cannot find Sybase codes."

      (* Connects to the specifed Sybase system using Kyle's interface. *)
      val (RepStr, ReqStr) = CIO.execute (!SybCodes,
            [!SybInterface, User, Passwd, Source])

      (* Function to shutdown connection. *)
      fun ShutDown _ = (CIO.close_in RepStr; CIO.close_out ReqStr)

      (* Function to process requests *)
      fun Dispatch Req ReplyCh ReadyCh = (
            (* Send query to database *)
            CIO.output (
               ReqStr,
               "QT"^(StringUtil.stringTrans("\n", " ")(COString.Km Req))^"\n");
            (* Send reply and signal ready. *)
            COI.ReadFromStreamLazy ReplyCh ReadyCh RepStr)

      (* Check that driver started and connected OK. *)
      val _ = if COBool.Km (COI.ReadFromStream RepStr)
```

```
              then ()
              else raise Server.StartupFailed
   in (Dispatch, ShutDown)
   end
```

- AddSybaseThread *Name User Password Source Level*

  Create |*Level*| many connections to *Source*. Name all these connections *Name*. If *Level* is positive, also annotate these connections as optimizable SQL connections — the system assume that the SQL queries sent to these conenctions are in a special optimizable form. Here is the implementation.

```
fun AddSybaseThread Name User Password Source Level =
   if Level < 0
   then (* Full Sybase Transact-SQL thread. Dont fool with it. *)
         let val SvNo = Server.Add
                  Name
                  (SybaseGen User Password Source)
                  (TYI.ReadFromString "string")
                  (fn _ => TYI.ReadFromString "{''1}")
                  (~Level)
         in Server.ResetAllConnections SvNo;
            "Sybase server thread " ^ Name ^ " added."
         end
   else (* Restricted optimizable SQL is used. *)
         let val SvNo = Server.Add
                  Name
                  (SybaseGen User Password Source)
                  (TYI.ReadFromString "string")
                  (fn _ => TYI.ReadFromString "{''1}")
                  Level
         in Server.ResetAllConnections SvNo;
            Annotations.AddSQL SvNo;
            "Sybase server thread " ^ Name ^ " added."
         end
```

- AddSybaseThreadPNo

  The function AddSybaseThread is made available for CPL as a primitive **sybase-add**, so that we can connect to Sybase databases from CPL. Kleisli refers to it using the primitive number AddSybaseThreadPNo. Here is the registration:

```
val AddSybaseThreadPNo = Primitive.Add
   "sybase-add"
   (COFun.Mk (fn UPS =>
      let val Name = COString.Km (CORecord.Proj LName UPS)
          val User = COString.Km (CORecord.Proj LUser UPS)
          val Passwd = COString.Km (CORecord.Proj LPasswd UPS)
          val Source = COString.Km (CORecord.Proj LSource UPS)
          val Level = CONum.Km (CORecord.Proj LLevel UPS)
      in COString.Mk (AddSybaseThread Name User Passwd Source Level)
      end))
   (TYI.ReadFromString "(#name: string, #user: string, #password: string, #server: string, #level: nu
```

- DeleteSybaseThread *Name*

  Remove all Sybase connections named *Name*. Implemented as

```
fun DeleteSybaseThread Name = (case Server.Peek Name
   of SOME SvNo => (
      Server.Delete SvNo;
      "Sybase thread " ^ Name ^ " deleted.")
   | NONE => "Sybase thread " ^ Name ^ " does not exist.")
```

- DeleteSybaseThreadPNo

  Make DeleteSybaseThread available to CPL as a primitive **sybase-del**. Kleisli refers to it using the primitive number DeleteSybaseThreadPNo. Here is the registration:

```
val DeleteSybaseThreadPNo = Primitive.Add
   "sybase-del"
   (COFun.Mk (COString.Mk o DeleteSybaseThread o COString.Km))
   (TYI.ReadFromString "string -> string")
```

- SybaseConnect *Name User Password Source Level*

  Make *Level* connections to the Sybase *Source*. Name all this conenctions *Name*. Return a pair of functions (*XactThread*, *ShutdownThread*). Send SQL query to any of these connection by calling *XactThread*(*query*). Shutdown all these connections by calling *ShutdownThread*().

- SybaseConnectPNo

  The fucntion SybaseConnect is made available to CPL as a primitive **sybase-connect**. Kleisli refers to it using the primitive number SybaseConnectPNo. Here is the registration:

```
val SybaseConnectPNo = Primitive.Add
   "sybase-connect"
   (COFun.Mk (fn UPS =>
      let val Name = COString.Km (CORecord.Proj LName UPS)
          val User = COString.Km (CORecord.Proj LUser UPS)
          val Passwd = COString.Km (CORecord.Proj LPasswd UPS)
          val Source = COString.Km (CORecord.Proj LSource UPS)
          val Level = CONum.Km (CORecord.Proj LLevel UPS)
          val (Dispatch, Shutdown) = SybaseConnect Name User Passwd Source Level
      in CORecord.MkPair (COFun.Mk Dispatch) (COFun.Mk Shutdown)
     end))
   (TYI.ReadFromString "(#name:string, #user:string, #password: string, #server:
 string, #level: num) -> (string -> {’’1}, ’2 -> unit)")
```

- SybaseXact *Name User Password Source Level Xact*

  Connect to Sybase using SybaseConnect *Name User Password Source Level*. Assume *Xact* is a transaction function. Apply it to the *XactThread* returned by SybaseConnect. Return the result of this transaction. Then call *ShutdownThread* to disconnect. Here is the implementation:

```
fun SybaseXact Name User Password Source Level Xact =
   let val (Dispatch, Shutdown) =  SybaseConnect Name User Password Source Level
   in Xact Dispatch before Shutdown ()
   end
```

- SybaseXactPNo

  The function SybaseXact is made available to CPL as the primitive **sybase-xact**. It is referred to be Kleisli using the primitive number SybaseXactPNo. Here is the registration:

```
val SybaseXactPNo = Primitive.Add
   "sybase-xact"
   (COFun.Mk (fn UPS =>
      let val Name = COString.Km (CORecord.Proj LName UPS)
          val User = COString.Km (CORecord.Proj LUser UPS)
          val Passwd = COString.Km (CORecord.Proj LPasswd UPS)
          val Source = COString.Km (CORecord.Proj LSource UPS)
          val Level = CONum.Km (CORecord.Proj LLevel UPS)
      in COFun.Mk (fn Xact =>
           let val Xact = (COFun.Km Xact) o COFun.Mk
           in SybaseXact Name User Passwd Source Level Xact
           end)
      end))
   (TYI.ReadFromString "(#name:string, #user:string, #password:string, #server:
string, #level: num) -> ((string -> {''1}) -> '2) -> '2")
```

83.4 The Restricted Optimizable Form of SQL

A partial grammar for the restricted optimizable form of SQL is given below. Capitalized terms are rule names. Literals are enclosed in quotes.

```
TOP
  : UNIONQUERY

UNIONQUERY
  : QUERY
  | "(" UNIONQUERY ")"
  | UNIONQUERY "union" UNIONQUERY

QUERY
  : "select" FIELDS "from" TABLES "where" CONDS
  | "select distinct" FIELDS "from" TABLES "where" CONDS
  | "select * from" TABLE "where" CONDS

FIELDS
  : FIELD
  | FIELD "," FIELDS

FIELD
  : COLUMN_NAME "=" REF

TABLES
  : TABLE
  | TABLE "," TABLES
```

```
TABLE
  : TABLE_NAME RANGE_VARIABLE
  | DATABASE_NAME ".." TABLE_NAME RANGE_VARIABLE

CONDS
  : COND
  | "(" COND ")"
  | "not" CONDS
  | CONDS "and" CONDS
  | CONDS "or" CONDS

COND
  : REF "=" REF
  | REF "<=" REF
  | REF "<" REF
  | REF ">=" REF
  | REF ">" REF
  | REF "like" REF
  | REF "in" "(" QUERY ")"
  | "exists" "(" QUERY ")"

REF
  : RANGE_VARIABLE "." COLUMN_NAME
  | REF OP REF        /* OP is a recognized SQL operators such as + and -. */
  | BASE              /* BASE is any recognized SQL atoms such as integers. */
  | OP "(" REFS ")"   /* OP is a recognized SQL function such as substring. */

REFS
  : REF
  | REF "," REFS
```

83.5 The Underlying C Program Sybcpl

A short manpage for the C program `sybcpl` is given below. You will need to know a bit about Sybase to understand it.

```
NAME
     sybcpl - Sybase retrieval using the Collection Programming Language (CPL).

SYNOPSIS
     sybcpl interface-file user password server

DESCRIPTION
     Sybcpl is a program for querying Sybase relational databases.
     It accepts on its standard input SQL commands. It forwards
     these commands to the specified server for execution.
     The results are then printed on its standard output using
     the compressed CPL Standard Exchange Format.
```

SQL commands are separated by new lines is prefixed by a two-letter code. The code for a normal query is "QT". Each command is forwarded as soon as they appear on the input.

OPTIONS

interface-file is the usual Sybase interface.
user is a authroized user the the target database.
password is the password of user.
server is the server for the target database.

AUTHOR

Jonathan Crabtree. crabtree@cbil.humgen.upenn.edu

84 SYSCALL

84.1 Synopsis

structure SYSCALL

This module implements calls to the Unix system. It offers a simple and flexible means to making use of existing facilities.

84.2 Signature

```
type svno = int (* server number *)

val coSv : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val coSvNo : svno
val lineSv : unit -> (CO.CO -> /[string]/ CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val lineSvNo : svno
val stringSv : unit -> (CO.CO -> /string/ CML.chan -> unit CML.chan -> unit) * ('a -> unit)
val stringSvNo : svno
```

84.3 Description

- coSv ()

  Create a system call server. Return a pair of functions (*Dispatch*, *Shutdown*). You can shut down the server by calling *Shutdown*. Or you can make a system call using *Dispatch Req ReplyCh ReadyCh*, where *Req* is the system call request, *ReplyCh* is the channel to which the result of the system call is sent and *ReadyCh* is the channel to which the ready signal from the server is sent. *Req* is expected to be a complex object triple (*Cmd*, *Args*, *Input*), where *Cmd* is a string—the Unix command—and *Args* is a list of strings—the command line arguments—and *Input* is a string to be used as input to the Unix command.

  Caution. You have to supply the full path of the Unix command. Also, the command must output result in the CPL Standard Exchange Format.

- coSvNo

  A system call server has been registered for CPL. Kleisli refers to it using the server number coSvNo. We also annotated it as a local server.

- lineSv ()

  Analogous to coSv. However, the result of system call does not need to be in the CPL Standard Exchange Format. It is returned as a list of string (one string for each line).

- lineSvNo

  Analogous to coSvNo. However, uses the server created by lineSv rather than coSv.

- stringSv ()

  Analogous to coSv. However, the result of system call does not need to be in the CPL Standard Exchange Format. It is returned as a long string.

- stringSvNo

  Analogous to coSvNo. However it uses the server created by stringSv rather than coSv.

85 TAXONSVR

85.1 Synopsis

structure TAXONSVR

Steve Brenner passed me a driver for NCBI's taxonomy database when he visited me in July 1997. The taxonomy is organized as a tree (what else?!) Each node of is identified by an id such as 3750 that is unique to the tree. Associated with each node are: a set of names—these are the scientific and common names such as "apples" and are not necessarily unique; a rank such as "species"; a parent node id such as 3749, and a set of children. (Question: who is the parent of the root of the taxonmy tree? Answer: that's the root itself!) The interface to Steve's driver is in the module TAXONSVR. Here are the functions provided by the interface.

85.2 Signature

```
val NameIdLab : labno
val IdInfoLab : labno
val LsIdLab   : labno
val LsNameLab : labno

val PhyCodes : string ref
val PhyDir : string ref
val DeletePhyThreadPNo : int
val DeletePhyThread : string -> string
val AddPhyThreadPNo : int
val AddPhyThread : string -> string
val PhyGen : unit -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)
```

85.3 Description

- NameIdLab

  The label number for #*nameid*.

- IdInfoLab

  The label number for #*idinfo*.

- LsIdLab

  The label number for #*lsid*.

- LsNameLab

  The label number for #*lsname*.

- PhyCodes

  The path to Brenner's driver for the taxonomy database. It is an alias of SVRCONFIG.PhyCodes.

- PhyDir

  The path to the NCBI taxonomy database. It is an alias of SVRCONFIG.PhyDir.

- DeletePhyThread *Name*

  Remove all connections named *Name* to the taxonomy database driver.

- DeletePhyThreadPNo

  The DeletePhyThread function is made available to CPL as the primitive `phy-del`. Kleisli refers to it using the primitive number DeletePhyThreadPNo.

- AddPhyThread *Name*

  Make a connection to the taxonomy driver and name the connection *Name*. Requests to the taxonomy driver can then be sent to this connection.

- AddPhyThreadPNo

  The AddPhyThread function is made available in CPL as the primitive `phy-add`. Kleisli refers to it using the primitive number AddPhyThreadPNo.

- PhyGen ()

  Make a connection to the taxonomy driver. Return a pair of function (*Dispatch*, *Shutdown*). Call *Shutdown*() to disconnect. Call *Dispatch Req ReplyCh ReadyCh* to make taxonomy query requests. Here *Req* is a complex object representing a taxonomy query, *ReplyCh* is the channel to which the reply from the taxonomy driver should be sent, and *ReadyCh* is the channel where the driver should signal ready. *Req* can take four possible forms, corresponding to the four different kind of services provided by the taxonomy driver. These forms are:

  - COVariant.Mk LsNameLab COUnit.Mk.
    This requests the taxonomy driver to return all the names in the taxonomy.
  - COVariant.Mk LsIdLab COUnit.Mk.
    This requests the taxonomy driver to return all the node id's in the taxonomy.
  - COVariant.Mk NameIdLab(COString.Mk *Name*).
    This requests the taxonomy driver to return the node id's corresponding to the *Name*.
  - COVariant.Mk IdInfoLab(CONum.Mk *Id*).
    This requests the taxonomy driver to return the information stored at the node identified by *Id*.

85.4 See Also

See also the *Collection Programming Language Reference Manual* [24].

86 WEBBLAST

86.1 Synopsis

structure WEBBLAST

The National Center for Biotechnology Information provides a web-based service for rapid and sensitive similarity searches against huge databases curated over there. It provides the same type of programs—`blastp`, `blastn`, `blastx`, `tblastn`, `tblastx`—similar to (but slightly different from) those in WU-BLAST2 described earlier in Section LOCALBLAST.

The driver for this web-based service consists of two Perl programs, `webblast` and `webblastdetail`, which return summary of BLAST hits and full detail of BLAST hits respectively. This module interfaces to the driver for this web-based service.

86.2 Signature

```
val DBLab : lno
val LevelLab : lno
val NameLab : lno

val BlastSummary : string ref
val BlastDetail : string ref
val SummaryType : string
val DetailType : string

val AddDoit : string -> string ref -> string -> string -> string -> int -> string
val AddDoitPNo : string -> string ref -> string -> string -> int
val DoitGen : string ref -> string -> string -> unit
      -> (CO.CO -> CO.CO CML.chan -> unit CML.chan -> unit) * ('a -> unit)

val TBLASTXSummaryPNo : pno
val BLASTXSummaryPNo : pno
val BLASTPSummaryPNo : pno
val BLASTNSummaryPNo : pno
val TBLASTNSummaryPNo : pno

val TBLASTNDetailPNo : pno
val TBLASTXDetailPNo : pno
val BLASTXDetailPNo : pno
val BLASTPDetailPNo : pno
val BLASTNDetailPNo : pno
```

86.3 Description

- DBLab

The label number for #*db*.

- LevelLab

  The label number for #*level*.

- NameLab

  The label number for #*name*.

- BlastSummary

  The path to the `webblast` program.

- BlastDetail

  The path to the `webblastdetail` program.

- SummaryType

  The CPL type of output produced by `webblast`. It looks like this: `{(#uid: num, #accession: string, #title: string, #score: num, #p-n: num, #n: num)}`.

- DetailType

  The CPL type of output produced by `webblastdetail`. It looks like this: `{(#uid: num, #accession: string, #title: string, #hits: [(#score: num, #pscore: num, #identities: num, #positives: num, #querystart: num, #queryend: num, #subjectstart: num, #subjectend: num, #alignment: string)])}`.

- TBLASTXSummaryPNo

  A function to make connections to the `tblastx` program at the National Center for Biotechnology Information and return hit summary has been registered for CPL as `webblast-tblastx`. Kleisli refers to it using the primitive number TBLASTXSummaryPNo.

- BLASTXSummaryPNo

  Ditto, but for `blastx`.

- BLASTPSummaryPNo

  Ditto, but for `blastp`.

- BLASTNSummaryPNo

  Ditto, but for `blastn`.

- TBLASTNSummaryPNo

  Ditto, but for `tblastn`.

- TBLASTNDetailPNo

  A function to make connections to the `tblastn` program in National Center for Biotechnology Information and return full hit detail has been registered for CPL as `webblast-tblastn-detail`. Kleisli refers to it using the primitive number TBLASTNDetailPNo.

- TBLASTXDetailPNo

  Ditto, but for `tblastx`.
- BLASTXDetailPNo

  Ditto, but for `blastx`.
- BLASTPDetailPNo

  Ditto, but for `blastp`.
- BLASTNDetailPNo

  Ditto, but for `blastn`.
- AddDoit
- AddDoitPNo
- DoitGen

86.4 See Also

See also LOCALBLAST. See also the *Collection Programming Language Reference Manual* [24].

87 WEBORF

87.1 Synopsis

structure WEBORF

The Virtual Genome Centre has a program for identifying open reading frames in DNA sequences. The method is based on [22]. The program is at `http://alces.med.umn.edu/webtrans.html`. Dapha Strauss developed a driver called `weborf` to this program. The interface to Daphna's program is in the module WEBORF.

87.2 Signature

87.3 Description

Part VI

Optimizers

88 Introduction

89 OPTCONFIG

89.1 Synopsis

structure OPTCONFIG

This modules puts various optimizer control flags in one place.

89.2 Signature

```
type 'a writer (* same as the 'a writer in CPLO *)

val InitWriter : (unit -> unit writer) ref
val Writer : (unit writer) ref
val DB :  (RULEDB.RULEDB * (RULEDB.RULEDB -> WellDone.SYN -> WellDone.SYN)) list ref
val DoOptimize : bool ref

val ShowIntermediateCodes : bool ref
val ShowFinalCodes : bool ref
```

89.3 Description

- InitWriter

  Store a function for creating a writer for writing optimization trace. This is an alias to RULEDB.Init. You must explicitly assign the writer thus created to be used as the default for writing optimization trace.
- Writer

  Store the writer currently used for writing optimization trace. This is an alias to RULEDB.Writer.
- DB

  Store the list of optimization phases. An optimization phase consists of a rule base and a rule application strategy. This is an alias to RULEDB.DB.
- DoOptimize

  Set this flag to true if you want to do optimization. This is an alias to RULEDB.DoOptimize.
- ShowIntermediateCodes

  Set this flag to true if you want all intermediate optimization steps to be traced. This is an alias to SimpleOptimizer.ShowIntermediateCodes.
- ShowFinalCodes

  Set this flag to true if you want to see the final optimized program. This is an alias to Interpretor.ShowFinalCodes.

89.4 See Also

See also RULEDB, SimpleOptimizer, and Interpretor.

For more information, consult also the *Kleisli/CPL Extensible Query Optimizer Programmer Guide* [26].

90 RULEDB

90.1 Synopsis

structure RULEDB

This module gives the definition of rules, rule bases, and strategies.

90.2 Signature

```
type 'a writer (* Same as the 'a writer in CPLO *)

type RULE = WellDone.SYN -> WellDone.SYN option

type RULEDB : {
   DoTrace : bool ref,
   Trace : (string -> WellDone.SYN -> WellDone.SYN -> unit) ref,
   Rules : : (string * RULE) list ref}

val Writer : (unit writer) ref
val Init : (unit -> unit writer) ref
val DoOptimize : bool ref
val Trace :  string -> WellDone.SYN -> WellDone.SYN -> unit
val Optimize : WellDone.SYN -> WellDone.SYN
val DB : (RULEDB * (RULEDB -> WellDone.SYN -> WellDone.SYN)) list ref
```

90.3 Description

- RULE

  The definition of a rewrite rule. Basically, a rewrite rule R is a function such that $R(E) = SOME(E')$ if E can be rewritten to E' using R.

- RULEDB

  The definition of a rule base. The DoTrace field is set to true if you want tracing to be done. The Trace field stores the function to be used for doing the tracing. The Rules field stores the rules and their names.

- Writer

  Store the writer to be used for writing the optimization trace. It is set to CPLO.OpenWriterOnStream CIO.std_err by default.

- Init

  Store the function for creating the writer for writing optimization trace. It is set to the function that returns CPLO.OpenWriterOnStream CIO.std_err by default.

- Trace N E E'

  This is the default function for writing out an optimization trace. It assumes E is rewritten to E' using the rule whose name is N.

- DB

  Store the optimization phases used by the optimizer. An optimization phase consists of a rule base and a rule application strategy. A rule application strategy is simple a function from WellDone.SYN to WellDone.SYN.

- DoOptimize

  Set this flag to true if you want to do optimization.

- Optimize N

  Optimize the query N by carrying out the optimization phases stored in DB. Return the optimized query.

90.4 See Also

For more information, consult also the *Kleisli/CPL Extensible Query Optimizer Programmer Guide* [26].

91 SimpleOptimizer

91.1 Synopsis

structure SimpleOptimizer

This module implements several useful rule application strategies for the optimizer.

91.2 Signature

```
type RULE = WellDone.SYN -> WellDone.SYN option
type STRATEGY = WellDone.SYN -> WellDone.SYN
type RULEDB : {
   DoTrace : bool ref,
   Trace : (string -> WellDone.SYN -> WellDone.SYN -> unit) ref,
   Rules : : (string * RULE) list ref}

val ShowIntermediateCodes : bool ref

val Pick : RULEDB -> RULE
val Decompose : STRATEGY -> STRATEGY
val ContextSensitive : RULE -> (RULEDB -> STRATEGY) -> RULE

val TopDownMany : RULEDB -> STRATEGY
val TopDownOnce : RULEDB -> STRATEGY
val BottomUpMany : RULEDB -> STRATEGY
val BottomUpOnce : RULEDB -> STRATEGY
val EntireOnce : RULEDB -> STRATEGY
val MaxOnce : RULEDB -> STRATEGY
val MaxMany : RULEDB -> STRATEGY
```

91.3 Description

- ShowIntermediateCodes

  Set this flag to true if you want to tracing intermediate rewrite steps.
- Pick D E

  Try the rules in D on E. Return `NONE` if no applicable rule can be found. Return `SOME` E' if some rule can be applied to rewrite E to E'.
- Decompose S E

  Rewrite all immediate subcomponents of E using S.
- TopDownMany D E

  Treat the query E as a tree. Repeatedly traverse and rewrite it from root to leaves using the rules in D.
- TopDownOnce D E

  Treat the query E as a tree. Traverse it once from root to leaves. Try to rewrite every node once using the rules in D. This and other strategies are generated using Pick and Decompose in a simple way like this:

```
fun TopDownOnce RDB N =
   let val Changed = ref false
       fun Pass N = Decompose Pass (case Pick RDB N
            of SOME N'=> (Changed := true; N')
            | NONE=> N)
       val N' = Pass (SyntaxTool1.Rename N)
       val Writer = !RULEDB.Writer
       fun Show P = (CPL0.WriteString Writer "\n\n---Current codes---\n";
                     CPL0.Write Writer P)
   in if !ShowIntermediateCodes andalso !Changed then Show N' else (); N'
   end
```

- BottomUpMany D E

  Treat the query E as a tree. Repeatedly traverse and rewrite it from leaves to root using the rules in D.

- BottomUpOnce D E

  Treat the query E as a tree. Traverse it once from leaves to root. Try to rewrite every node once using the rules in D.

- EntireOnce D E

  Apply all the rules in D to the whole of E. That is, apply the rules at the root of E.

- MaxOnce D E

  Treat the query E as a tree. Traverse the tree from root to leaves. Try to apply the rules in D to every traversal path once at a node that is as close to the root as possible. In other words, rewrite maximal applicable subexpressions of E only.

- MaxMany D E

  Repeatedly rewrite E using the rules in D by applying the MaxOnce strategy.

- ContextSensitive R S E

  Apply the rule R to query E using strategy S. Implemented as

```
fun ContextSensitive Rule Strategy Query =
   let (* This flag is set if Rule is applied *)
      val Changed = ref false
      (* Set up a context-sensitive rule base *)
      val RDB = {
         DoTrace = ref true,
         Trace = ref (fn _ => fn _ => fn _ => Changed := true),
         Rules = ref [("", Rule)]}
      (* Apply rule using strategy *)
      val Query' = Strategy RDB Query
   in if !Changed
      then SOME Query'
      else NONE
   end
```

91.4 See Also

For more information, consult also the *Kleisli/CPL Extensible Query Optimizer Programmer Guide* [26].

Part VII

Tools for SQL

Part VIII

Support Environments

92 Introduction

References

[1] J. Albert. Algebraic properties of bag data types. In *Proceedings of 17th International Conference on Very Large Databases*, pages 211–219, 1991.

[2] S. F. Altschul and W. Gish. Local alignment statistics. *Methods in Enzymology*, 266:460–480, 1996.

[3] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[4] V. Breazu-Tannen, P. Buneman, and S. Naqvi. Structural recursion as a query language. In *Proceedings of 3rd International Workshop on Database Programming Languages, Naphlion, Greece*, pages 9–19. Morgan Kaufmann, August 1991. Also available as UPenn Technical Report MS-CIS-92-17.

[5] V. Breazu-Tannen and R. Subrahmanyam. Logical and computational aspects of programming with Sets/Bags/Lists. In *LNCS 510: Proceedings of 18th International Colloquium on Automata, Languages, and Programming, Madrid, Spain, July 1991*, pages 60–75. Springer Verlag, 1991.

[6] Peter Buneman, Shamim Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types. *Theoretical Computer Science*, 149(1):3–48, September 1995.

[7] Anthony J. Field and Peter G. Harrison. *Functional Programming.* Addison-Wesley, Wokingham, England, 1988.

[8] Stephane Grumbach and Tova Milo. Towards tractable algebras for bags. In *Proceedings of 12th ACM Symposium on Principles of Database Systems*, pages 49–58, Washington, D. C., May 1993.

[9] Carl A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* Foundations of Computing. MIT Press, 1992.

[10] Richard Hull and Chee K. Yap. The Format model: A theory of database organisation. *Journal of the ACM*, 31(3):518–537, July 1984.

[11] S. Karlin and S. F. Altschul. Applications and statistics for multiple high-scoring segments in molecular sequences. *Proc. Natl. Acad. Sci. USA*, 90:5873–5877, 1993.

[12] Leonid Libkin and Limsoon Wong. Some properties of query languages for bags. In Catriel Beeri, Atsushi Ohori, and Dennis E. Shasha, editors, *Proceedings of 4th International Workshop on Database Programming Languages, New York, August 1993*, pages 97–114. Springer-Verlag, January 1994. See also UPenn Technical Report MS-CIS-93-36.

[13] Leonid Libkin and Limsoon Wong. Query languages for bags and aggregate functions. Technical Report 95-167-0, Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, January 1995.

[14] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML.* MIT Press, 1990.

[15] A. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: A structural classification of protein database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[16] National Center for Biotechnology Information, National Library of Medicine, Bethesda, MD. *ENTREZ: Sequences Users' Guide*, 1992. Release 1.0.

[17] National Center for Biotechnology Information, National Library of Medicine, Bethesda, MD. *NCBI ASN.1 Specification*, 1992. Revision 2.0.

[18] Atsushi Ohori. A compilation method for ML-style polymorphic record calculi. In A. W. Appel, editor, *Proceedings of 19th ACM Symposium on Principles of Programming Languages*, pages 154–165, 1992.

[19] L. C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1991.

[20] Marian Petre. Expert programmers and programming languages. In J. M. Hoc, T. R. G. Green, R. Samurcay, and D. J. Gilmore, editors, *Psychology of Programming*, chapter 2.1, pages 103–115. Academic Press, London, England, 1989.

[21] Didier Remy. Efficient representation of extensible records. In P. Lee, editor, *Proceedings of ACM SIGPLAN Workshop on ML and its Applications*, pages 12–16, 1992.

[22] P. M. Sharp and W. H. Li. The codon adaptation index–a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, 15(3):1281–1295, February 1987.

[23] Limsoon Wong. *Querying Nested Collections*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, August 1994. Available as University of Pennsylvania IRCS Report 94-09.

[24] Limsoon Wong. *The Collection Programming Language Reference Manual*. Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, October 1995. Available at `http://sdmc.iss.nus.sg/kleisli/psZ/cpl-defn.ps`.

[25] Limsoon Wong. An introduction to Remy's fast polymorphic projection. *ACM SIGMOD Record*, 24(3):34–39, September 1995.

[26] Limsoon Wong. *The Kleisli/CPL Extensible Query Optimizer Programmer Guide*. Institute of Systems Science, Heng Mui Keng Terrace, Singapore 119597, November 1995. Available at `http://sdmc.iss.nus.sg/kleisli/psZ/cplopt.ps`.

[27] Limsoon Wong. Normal forms and conservative extension properties for query languages over collection types. *Journal of Computer and System Sciences*, 52(3):495–505, June 1996.

[28] John Wootton and Scott Federhen. Statistics of local complexity in amino acid sequences and sequence databases. *Computers and Chemistry*, 17:149–163, 1993. Talks about the NCBI seg protein filter software.