Contents

1 Module `Support` : Collects a number of low-level facilities used by the other modules in the typechecker/evaluator.

```
module Pervasive :
  sig
    val pr : string -> unit
  end
```

Some pervasive abbreviations – opened everywhere by convention

```
module Error :
  sig
    exception Exit of int
```

An exception raised by the low-level error printer; exported here so that it can be caught in module Main and converted into an exit status for the whole program.

```
    type info
```

An element of the type info represents a "file position": a file name, line number, and character position within the line. Used for printing error messages.
An element of the type info represents a UNKNOWN file position

```
    val dummyinfo : info
    val createInfo : string -> int -> int -> info
```

Create file position info: filename lineno column

```
    val printInfo : info -> unit
    type 'a withinfo = {
      i : info ;
      v : 'a ;
    }
```

A convenient datatype for a "value with file info." Used in the lexer and parser.

```
    val errf : (unit -> unit) -> 'a
```

Print an error message and fail. The printing function is called in a context where the formatter is processing an hvbox. Insert calls to Format.print_space to print a space or, if necessary, break the line at that point.

Adds fileinfo to the printing function

```
val errfAt : info -> (unit -> unit) -> 'a
val err : string -> 'a
```

Convenient wrappers for the above, for the common case where the action to be performed is just to print a given string.

First step of the error chain. Adds the error string to the printing function

```
val error : info -> string -> 'a
val warning : string -> unit
```

Variants that print a message but do not fail afterwards

Variants that print a message but do not fail afterwards with fileinfo

```
val warningAt : info -> string -> unit
end
```

Error printing utilities – opened everywhere by convention

2 Module Syntax : Syntax trees and associated support functions

```
type term =
  | TmTrue of Support.Error.info
  | TmFalse of Support.Error.info
  | TmIf of Support.Error.info * term * term * term
  | TmVar of Support.Error.info * int * int
  | TmAbs of Support.Error.info * string * term
  | TmApp of Support.Error.info * term * term
  | TmRecord of Support.Error.info * (string * term) list
  | TmProj of Support.Error.info * term * string
  | TmFloat of Support.Error.info * float
  | TmTimesfloat of Support.Error.info * term * term
  | TmString of Support.Error.info * string
  | TmZero of Support.Error.info
  | TmSucc of Support.Error.info * term
  | TmPred of Support.Error.info * term
  | TmIsZero of Support.Error.info * term
  | TmLet of Support.Error.info * string * term * term
  | TmRec of Support.Error.info * string * term
```

All the supported terms in the language, including an info component describing its appearance on the source file if possible. Some terms are compound, including other terms.

```
val termTypeToString : term -> string
```

Auxiliary function to recover the name of the term of a given type

```
type binding =
  | NameBind
  | TmAbbBind of term
```

A binding is either a name (let ... in) or an abstraction (... = ...) over a term

```
type dbg =
  | DbgContextualize
  | DbgStartTrace
  | DbgEndTrace
```

Available types for the debug commands

```
val __TRACE__ : string
```

Trace flag

Binding to activate the tracing facility

```
val __TRACE_ON__ : binding
```

Binding to deactivate the tracing facility

```
val __TRACE_OFF__ : binding
```

Initial status of the trace flag (off)

```
val initialTraceFlag : string * binding
type command =
  | Eval of Support.Error.info * term
  | Bind of Support.Error.info * string * binding
  | Debug of Support.Error.info * dbg
```

A command is an order in the language, it can be either a binding, a term for evaluation or a debugging command

```
type context = (string * binding) list
```

A context is a list of tuples string * binding type, where, if the binding is an abstract binding, it includes a term.

```
val emptycontext : context
```

Provides an empty context

Returns the length of the specified context

```
val ctxlength : context -> int
```

Adds a binding to the given context

`val addbinding : context -> string -> binding -> context`

Adds a name in the current context

`val addname : context -> string -> context`

Given an index and a context, returns the variable name if it exists or raises an error otherwise

`val index2name : Support.Error.info -> context -> int -> string`

Given an index and a context, returns the binding associated to that index

`val getbinding : Support.Error.info -> context -> int -> binding`

Given a name and a context, returns its index on that context

`val name2index : Support.Error.info -> context -> string -> int`

Checks if a name is bound in the given context

`val isnamebound : context -> string -> bool`

`val termShift : int -> term -> term`

Displaces a variable term's indices by an specified amount

`val termSubstTop : term -> term -> term`

Substitutes every apparition of a variable inside the second term with the value passed as the first term

`val applyToFPC : context -> term -> term`

Applies the given term to the FPC allowing for recursion

`val printtm : context -> term -> unit`

Prints a term

Prints an atomic term (true, 9, zero...)

`val printtm_ATerm : bool -> context -> term -> unit`

Prints a binding's name and associated terms

`val prbinding : context -> binding -> unit`

`val tmInfo : term -> Support.Error.info`

Extracts term filename, line number and column information

3 Module `Core` : Core typechecking and evaluation functions

`val eval : Syntax.context -> Syntax.term -> Syntax.term`

Evaluation function: Provided with a context and a term, it evaluates that term or returns it unchanged if there's no evaluation rule

`val evalbinding : Syntax.context -> Syntax.binding -> Syntax.binding`

Evaluation function for bindings. Evaluates a binding and modifies the given context to include it.

`val print_context : Syntax.context -> unit`

Output for the current context

This function enable o disable trace printing

`val check_trace : Syntax.context -> bool`

Evaluation function for debugging commands. Evaluates a debugging command and modifies the given context to include it.

`val debugging : Syntax.context -> Syntax.dbg -> Syntax.context`