A "Safe" Sockets API

John Reppy
AT&T Bell Laboratories
Sept 20, 1995

Background

The BSD sockets interface unifies a collection of different network protocols into a single socket type and address type.

```
type   sock
type   sock_addr

val    socket  : (addr_family * sock_type)
                 -> sock

val    accept  : sock -> (sock * sock_addr)
val    bind    : (sock * sock_addr) -> unit
val    connect : (sock * sock_addr) -> unit
val    listen  : (sock * int) -> unit
val    close   : sock -> unit
```

Address Family	Socket type: DGRAM	STREAM
INET	UDP	TCP
UNIX	Unix DGRAM	Unix Streams (Pipes)

Not all operations work on all kinds of sockets. Plus, the socket kind and ~~socket~~ network address must be consistent.

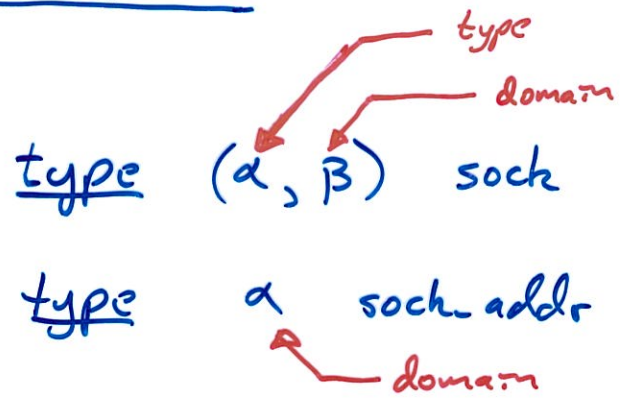
How can we use the type system to protect against ~~misusing~~ ~~use~~ confusing different kinds of sockets?

First attempt: use the module system.

```
signature SOCK =
  sig
    type sock
    type sock_addr

    val socket : unit -> sock

    val connect : (sock * sock_addr) -> unit
      :
  end

structure TCPSock : sig
    include SOCK
    val accept : sock -> (sock * sock_addr)
    val listen : (sock * int) -> unit
      :
end = struct ... end

structure UDPSock : sig
    include SOCK
      :
end = struct ... end
```

Problem: "polymorphic" uses of sockets not allowed

A better solution:



Introduce void types to constrain the polymorphism

```
type  stream  }
type  dgram   }  socket types
type  ip      }
type  unix    }  address families
```

```
val accept : (stream, α) sock
             → ((stream, α) sock × α sock_addr)

val listen : ((stream, α) sock × int) → unit

val bind : ((α, β) sock × β sock_addr) → unit
  ⋮
```

Socket creation:

```
structure IPSock : sig

    val addr : (inet_addr * int) -> ip sock_addr

    val udpSocket : unit -> (dgram, ip) sock
    val tcpSocket : unit -> (stream, ip) sock

end = struct ... end
```