

$\text{phi}>$ The Applied π -calculus Interpreter

Will de Renzy-Martin

June 23, 2014



Handshake Protocol in Plain English

- ▶ RFC Specification



Handshake Protocol in phi

```
let clientA(pkA,skA,pkB) =  
  (out(c,pkA); in(c,x);  
   let s = "Secret Message" in  
   let y = adec(skA,x) in  
   let pair(pkD,j) = getmsg(y) in if pkD = pkB then  
                                   let k = j in  
   out(c,senc(k,s)))  
let serverB(pkB,skB) =  
  in(c',pkX);  
  new k;  
  out(c',aenc(pkX,sign(pkB,pair(pkB,k))));  
  in(c',x);  
  let z = sdec(k,x) in out(stdout,z)
```



Process Calculi



Process Calculi

- Communication



Process Calculi

- ▶ Communication
- ▶ Sequential Composition



Process Calculi

- ▶ Communication
- ▶ Sequential Composition
- ▶ Parallel Composition



Process Calculi

- ▶ Communication
- ▶ Sequential Composition
- ▶ Parallel Composition
- ▶ Reduction Semantics



Process Calculi

- ▶ Communication
- ▶ Sequential Composition
- ▶ Parallel Composition
- ▶ Reduction Semantics
- ▶ Hiding



Process Calculi

- ▶ Communication
- ▶ Sequential Composition
- ▶ Parallel Composition
- ▶ Reduction Semantics
- ▶ Hiding
- ▶ Recursion and Replication



Process Calculi

- ▶ Communication
- ▶ Sequential Composition
- ▶ Parallel Composition
- ▶ Reduction Semantics
- ▶ Hiding
- ▶ Recursion and Replication
- ▶ The Null Process



phi

0	Null Process
in(u,M)	Message in
out(u,M)	Message out
P Q	Parallel Composition
P ; Q	Sequential Composition
!P	Replication
new x	Name reservation
if p(M) then P else Q	Conditionals
let X = M in P	Hiding / Pattern Matching
&t	Function call



Processes

```
eval env (Conc procs) = do
  var <- liftIO newEmptyMVar
  mapM_ (forkProcess var) procs
  res <- liftIO (takeMVar var)
  case res of
    Left err  -> throwE err
    Right _   -> return ()
where
  forkProcess var proc = liftIO $ forkIO $ do
    res <- runExceptT (eval env proc)
    _ <- tryPutMVar var res
    return ()
```



Channels

```
data Channel = Channel {  
    send      :: String -> IO ()  
    , receive :: IO String  
    , extra   :: [String]  
}
```



Primitives

```
primitives :: [(Name      , TermFun)]
primitives = [ ("fst"      , first)
              , ("snd"      , secnd)
              , ("hash"     , hash)
              , ("getmsg"   , getmsg)
              , ("sdec"     , sdec)
              , ("senc"     , binaryId "senc")
              , ("adec"     , adec)
              , ("aenc"     , binaryId "aenc")
              , ("sign"     , binaryId "sign")
              , ("checksign", checksign)
              , ("mac"      , mac) ..]
```



Pattern Matching

```
let ls = list(1,pair(2,list(3,4,5)),6) in  
  let list(_1,pair(_2,list(_3,x,_5)),_6) = ls in  
    out(stdout,x)
```



Pattern Matching

```
let follow(ch,r) = (out(ch,r);in(ch,resp:HttpResponse);  
  let list(c,_,_,b) = resp in  
    if c = 302  
      then let req = httpReq(getHeader("location",resp),  
                               headers(),  
                               httpGet()) in  
        &follow(ch,req)  
    else out(stdout,b))
```



Limitations



Limitations

- ▶ "Single Shot" Socket Channels



Limitations

- ▶ "Single Shot" Socket Channels
- ▶ Cryptographically insecure



Limitations

- ▶ "Single Shot" Socket Channels
- ▶ Cryptographically insecure
- ▶ Hacks



Installing

The source is available on Hackage, and can be installed using cabal:

```
cabal update
cabal install pi-calculus
```

Alternatively you can clone the source and build using cabal:

```
git clone git@github:renzyq19/pi-calculus
cd pi-calculus/pi
cabal install
```



Demonstrations



Demonstrations

- ▶ Simple Client - Server



Demonstrations

- ▶ Simple Client - Server
- ▶ Secret Channel Client - Server



Demonstrations

- ▶ Simple Client - Server
- ▶ Secret Channel Client - Server
- ▶ Follow Redirects



Wrap Up



Wrap Up

- Questions?

